(2)

User's Guide
CMU/SEI-90-UG-1
ESD-90-TR-5

Carnegie-Mellon University
Software Engineering Institute

Hartstone Benchmark User's Guide, Version 1.0

Patrick Donohoe
Ruth Shapiro
Nelson Weiderman

March 1990

User's Guide
CMU/SEI-90-UG-1
ESD-90-TR-5
March 1990

Hartstone Benchmark User's Guide, Version 1.0

Patrick Donohoe
Ruth Shapiro
Nelson Weiderman
Real-Time Embedded Systems Testbed Project

Approved for public release.
Distribution unlimited.

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This technical report was prepared for the

SEI Joint Program Office
ESD/AVS
Hanscom AFB, MA 01731

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

Review and Approval

This report has been reviewed and is approved for publication.

FOR THE COMMANDER

[illegible]
Charles J. Ryan, Major, USAF
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense.

Table of Contents

List of Figures

Hartstone Benchmark User's Guide, Version 1.0

Abstract: The Hartstone benchmark is a set of timing requirements for testing a system's ability to handle hard real-time applications. It is specified as a set of processes with well-defined workloads and timing constraints. The name *Hartstone* derives from *HA*rd *R*eal *T*ime and the fact that the workloads are presently based on the well-known Whetstone benchmark. This report describes the structure and behavior of an implementation in the Ada programming language of one category of Hartstone requirements, the Periodic Harmonic (PH) Test Series. The Ada implementation of the PH series is aimed primarily at real-time embedded processors where the only executing code is benchmark and the Ada runtime system. Guidelines for performing various Harts ,ne experiments and interpreting the results are provided. Also included are the source code listings of the benchmark, information on how to obtain the source code in machine-readable form, and some sample results for Version 1.0 of the Systems Designers XD Ada VAX/VMS - MC68020 cross-compiler.

1. Introduction

The Hartstone benchmark comprises a series of requirements to be used for testing the ability of a system to handle hard real-time applications. Its name derives from Hard Real Time and the fact that the computational workload of the benchmark is provided by a variant of the Whetstone benchmark [Curnow 76], [Harbaugh 84], [Wichmann 88]. "Hard" real-time applications *must* meet their deadlines to satisfy system requirements; this contrasts with "soft" real-time applications where a statistical distribution of response times is acceptable [Liu 73]. The rationale and operational concept of the Hartstone benchmark are described in [Weiderman 89]; in particular, five test series of increasing complexity are defined and one of these, the Periodic Harmonic (PH) Test Series, is described in detail.[1]

This user's guide describes the design and implementation of the PH series in the Ada programming language [LRM 83]. The overall structure and behavior of the benchmark programs are described, implementation-dependent aspects of the design are noted, and guidelines for performing the experiments described in [Weiderman 89] and interpreting their results are provided. Source code for the benchmark and sample results for the Systems Designers XD Ada VAX/VMS to Motorola MC68020 cross-compiler, Version 1.0, are included as appendices, as well as information on how to obtain machine-readable copies of the Hartstone source code and supporting documentation.

This Ada implementation of the Hartstone PH test series is aimed primarily at real-time embedded or "bare-board" target systems. It is assumed that on such systems the only executing code is the Hartstone code and the Ada runtime system. Hartstone can be used to gauge the performance of the Ada runtime system and its ability to handle multiple real-time tasks efficiently. As this guide explains, Hartstone is not a simple benchmark that produces just one number

[1]This document is recommended reading for people wishing to gain a broader understanding of the issues that motivated the concept of the Hartstone benchmark.

representing the "score" of the runtime system. The output from all Hartstone experiments must be considered, as well as the characteristics of the target processor, when drawing conclusions based on Hartstone results.

2. Periodic Harmonic Test Series

2.1. Periodic Tasks

The Periodic Harmonic (PH) Test Series is the simplest of the five test series defined in [Weiderman 89] for the Hartstone benchmark. The Ada implementation (the "Delay/ND" design discussed in [Weiderman 89]) consists of a set of five periodic Ada tasks that are independent in the sense that their execution need not be synchronized; they do not communicate with each other. Each periodic task has a *frequency*, a *workload*, and a *priority*. Task frequencies are harmonic: the frequency of a task is an integral multiple of the frequency of any lower-frequency task. Frequencies are expressed in Hertz; the reciprocal of the frequency is a task's period, in seconds.

A task workload is a fixed amount of work, which must be completed within a task's period. The workload of a Hartstone periodic task is provided by a variant of the well-known composite synthetic Whetstone benchmark [Curnow 76] called Small_Whetstone [Wichmann 88]. Small_Whetstone has a main loop which executes one thousand Whetstone instructions, or one *Kilo-Whetstone*. A Hartstone task is required to execute a specific number of Kilo-Whetstones within its period. The rate at which it does this amount of work is measured in Kilo-Whetstone instructions per second, or *KWIPS*. This *workload rate*, or speed, of a task is equal to its per-period workload multiplied by the task's frequency. The *deadline* for completion of the workload is the next scheduled activation time of the task. Successful completion on time is defined as a *met deadline*. Failure to complete the workload on time results in a *missed deadline* for the task. Missing a deadline in a hard real-time application is normally considered a system failure. In the Hartstone benchmark, however, processing continues in order to gather additional information about the nature of the failure and the behavior of the benchmark after deadlines have begun to be missed. Therefore, in the Ada implementation of the PH series, if a task misses a deadline it attempts to compensate by not doing any more work until the start of a new period. This process, called *load-shedding*, means that if a deadline is missed by a large amount (more than one period, say) several work assignments may be cancelled. Deadlines ignored during load-shedding are known as *skipped deadlines*. The reason for load-shedding is that "resetting" offending tasks and letting the test series continue allows more useful information to be gathered about the failure pattern of the task set. The conditions under which the test series eventually completes are discussed in Section 2.2.

Task priorities are assigned to tasks according to a *rate-monotonic* scheduling discipline [Liu 73], [Sha 89]. This means that higher-frequency tasks are assigned a higher priority than lower-frequency tasks. The priorities are fixed and distinct. The rate-monotonic priority assignment is optimal in the sense that no other fixed-priority assignment scheme can schedule a task set that cannot be scheduled by the rate-monotonic scheme [Liu 73]. In the Hartstone task set, priorities are statically assigned at compile time via the Priority pragma. Task 1 has the lowest priority and task 5 has the highest. The main program which starts these tasks is assigned a priority higher than any task so that it can activate all tasks via an Ada rendezvous.

A task implements periodicity by successively adding its period to a predetermined starting time to compute its next activation time. Within a period, it does its workload and then suspends itself until its next activation time. This paradigm, based on the one shown in Section 9.6 of the *Ada Language Reference Manual* [LRM 83], was adopted because of its portability, portability being one of the major objectives of the Hartstone benchmark. The implications of using this paradigm are discussed in Section 5.4.

2.2. Hartstone Experiments

Four *experiments* have been defined for the PH series, each consisting of a number of *tests*. A test will either *succeed* by meeting all its deadlines, or *fail* by not meeting at least one deadline. The Hartstone main program initiates a test by activating the set of Hartstone tasks; these perform the actual test by executing their assigned workloads, periodically, for the duration of the test. A test will always run for its predefined test duration. When a test finishes, the results are collected by the main program and a check is made to see if the test results satisfy a user-defined *completion criterion* for the entire experiment. If they do, the experiment is over and a summary of the entire experiment is generated; if not, a new test is initiated and the experiment continues. Experiment completion criteria are defined later in this section.

Each new test in an experiment is derived from the characteristics of the preceding test. The first test, called the *baseline test*, is the same for all experiments: activate the initial set of Hartstone tasks (called the *baseline task set*) and collect the results from them. As an example, the baseline test below has a total workload rate of 320 Kilo-Whetstone instructions per second (KWIPS)[2] allocated as follows:

Task No.	Frequency (Hertz)	Kilo-Whets per period	Kilo-Whets per second
1	2.00	32	64.00
2	4.00	16	64.00
3	8.00	8	64.00
4	16.00	4	64.00
5	32.00	2	64.00

			320.00

[2]This baseline test is different from that of [Weiderman 89]; the frequencies and workloads have been doubled. This doubling was done initially to cause deadlines to be missed after fewer iterations, so that experiments would complete in a shorter time. The original task set proved to be too low a starting point for the cross-compiler and target used in Hartstone prototype testing, the Systems Designers XD Ada compiler, and a 12.5 MHz Motorola MC68020 target processor. During subsequent testing on a number of different cross-compilers, stronger reasons for increasing or decreasing the frequencies and workloads of the baseline task set emerged. A more detailed discussion of desirable properties of the baseline task set appears in Section 5.2.

The four experiments are:

Experiment 1: Starting with the baseline task set, the frequency of the highest frequency task (task 5) is increased for each new test until a task misses a deadline. The frequencies of the other tasks and the per-period workloads of all tasks do not change. The amount by which the frequency increases must preserve the harmonic nature of the task set frequencies: this means a minimum increase by an amount equal to the frequency of task 4. For the previous example, this sequence increases the task set's total workload rate by 32 KWIPS (16 Hertz, the frequency increment, times task 5's per-period workload) at a time and tests the system's ability to handle a fine granularity of time (the decreasing period of the highest-frequency task) and to switch rapidly between processes.

Experiment 2: Starting with the baseline task set, all the frequencies are scaled by 1.1, then 1.2, then 1.3, and so on for each new test until a deadline is missed. The per-period workloads of all tasks do not change. The scaling preserves the harmonic frequencies; it is equivalent to multiplying the frequencies of the current test by 0.1 to derive those of the next test. As with experiment 1, this sequence increases the total workload rate in the above example by 32 KWIPS. By contrast with experiment 1, the increasing rates of doing work affect all tasks, not just one.

Experiment 3: Starting with the baseline task set, the workload of each task is increased by 1 Kilo-Whetstone per period for each new test, continuing until a deadline is missed. The frequencies of all tasks do not change. This sequence increases the total workload rate in the example by 62 KWIPS at a time, without increasing the system overhead in the same way as in the preceding experiments.

Experiment 4: Starting with the baseline task set, new tasks with the same frequency and workload as the "middle" task, task 3, of the baseline set are added until a deadline is missed. The frequencies and workloads of the baseline task set do not change. This sequence increases the total workload rate in the example by 64 KWIPS at a time and tests the system's ability to handle a large number of tasks.

When the computational load, plus the overhead, required of the periodic tasks eventually exceeds the capability of the target system, they will start to miss their deadlines. An experiment is essentially over when a test misses at least one deadline. For the purpose of analysis, it may be useful to continue beyond that point; therefore, tests attempt to compensate for missed deadlines by shedding load, as described previously. A Hartstone user has the choice of stopping the experiment at the point where deadlines are first missed or at some later point. The *completion criteria* for an experiment are largely defined in terms of missed and skipped deadlines. An experiment completes when a test satisfies one of the following user-selected criteria:

- Any task in the task set misses at least one deadline in the current test.
- The cumulative number of missed and skipped deadlines for the task set, in the current test, reaches a pre-set limit.
- The cumulative number of missed and skipped deadlines for the task set, in the current test, reaches a pre-set percentage of the total number of deadlines. This criterion is an alternative to specifying an absolute number of missed and skipped deadlines.

- The workload required of the task set is greater than the workload achievable by the benchmark in the absence of tasking. This is a default completion criterion for all experiments.
- The default maximum number of extra tasks has been added to the task set and deadlines still have not been missed or skipped. This is a default completion criterion for experiment 4. If this happens, the user must increase the value of the parameter representing the maximum number of tasks to be added.

2.3. Overall Benchmark Structure and Behavior

The Ada implementation of the PH series consists of three Ada packages and a main program. A Booch-style diagram illustrating dependencies between these Hartstone units is shown in Figure 2-1. The arrows represent **with** clauses. The Workload package contains the Small_Whetstone procedure that provides the synthetic workload for Hartstone periodic tasks. The Periodic_Tasks package defines the baseline set of tasks, and a task type to be used in the experiment where new tasks are added to the baseline set. The Experiment package provides procedures to initialize experiments, get the characteristics of a new test, check for experiment completion, and store and output results. It also defines the frequencies and workloads to be assigned to the baseline task set, as well as the experiment completion criteria. Initialization of an experiment includes a "calibration" call to Small_Whetstone to measure the procedure's raw speed; this is why the dependency diagram shows a dependency of package Experiment on package Workload. The main Hartstone program controls the starting and stopping of tasks, and uses procedures provided by the Experiment package to output results of individual tests and a summary of the entire experiment.

The compilation order of the packages and main program is as follows:

```
package   Workload
package   Periodic_Tasks
package   Experiment
procedure Hartstone
```

Tasks obtain the starting time, duration, frequency, and workloads of the test from a rendezvous with the main Hartstone program and then proceed independently. On completion of a test, the results are collected by the main program in a second rendezvous, and may optionally be written at that point. The main program then starts the next test in the experiment and the experiment continues until it satisfies the user-defined completion criterion. On completion of the experiment, a summary of the entire experiment is generated. Details of the output produced by Hartstone tests are given in Section 5.1.

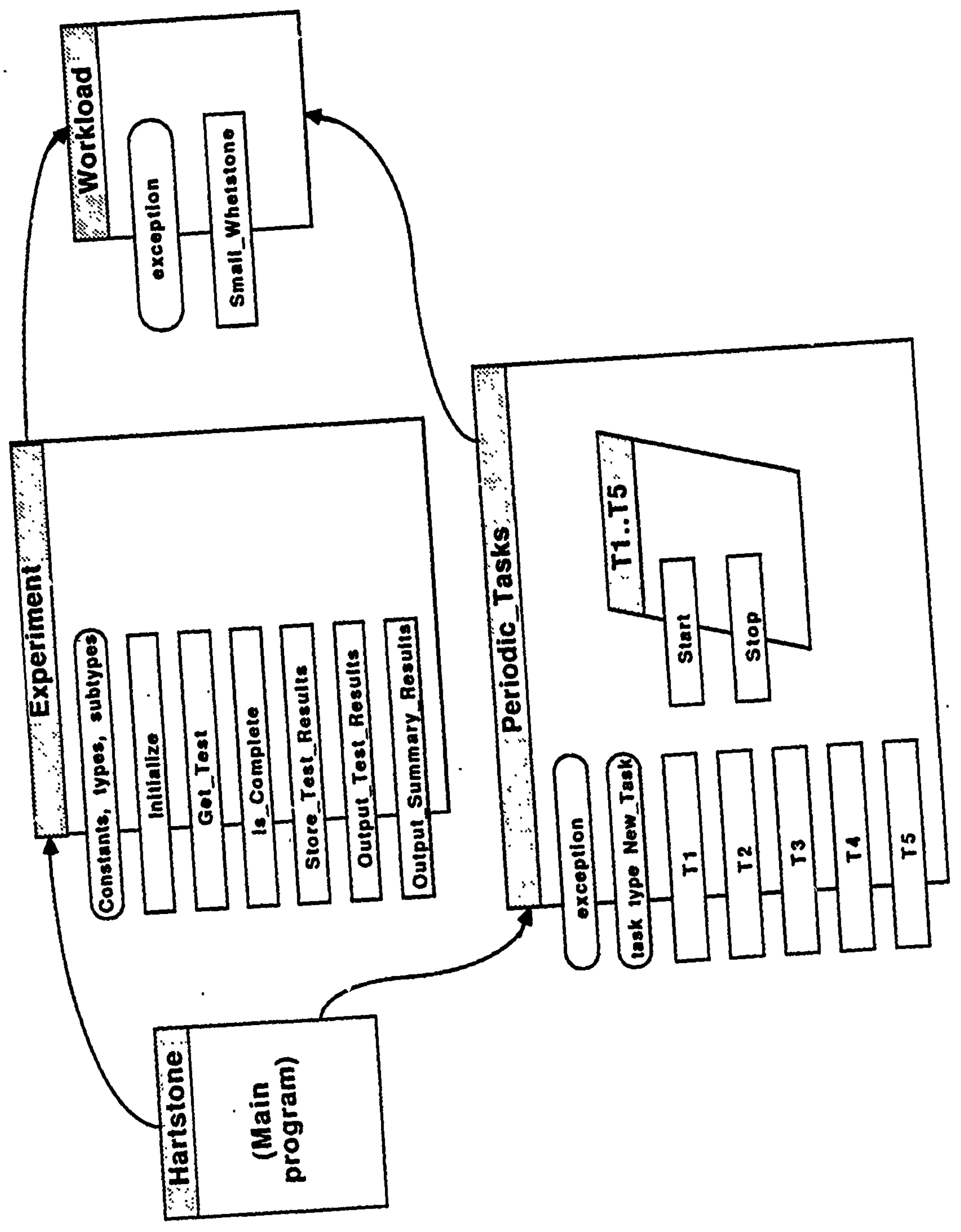

Figure 2-1: Hartstone Dependency Diagram

3. Hartstone Portability

The Ada version of the Hartstone benchmark for the PH series is written entirely in Ada and is intended to be portable over a wide range of Ada compilation systems. However, it does have certain implementation-dependent features which can be classified in two broad categories: features affecting the portability of the source code and features affecting the runtime performance of Hartstone executable code. The principal portability issues are Hartstone's use of mathematical library functions and predefined types. These also influence the performance, of course, but a discussion of performance factors will be deferred until Section 5.4.

Mathematical Libraries. The Small_Whetstone benchmark (and the full Whetsone benchmark, from which it is derived) performs computations involving transcendental functions; these functions are typically provided by a mathematical library package supplied with the Ada compilation system. The names used by vendors for mathematical libraries vary greatly, so a user will need to ensure that the correct library name for the system is being used in the **with** and **use** clauses in the body of package Workload wherein Small_Whetstone is encapsulated. Also, the names of some of the functions in these libraries may vary: for example, in some libraries, the natural logarithm function is named "Log," while for others it is named "Ln." An additional problem is caused by the fact that "Log" is used, in some libraries, to designate the base 10 logarithm function. The Small_Whetstone procedure requires the natural logarithm function for its calculations to be correct, so inadvertent use of a base 10 function will cause a runtime exception. This exception is typically either a Constraint_Error or an exception defined within Small_Whetstone that is raised when Small_Whetstone's internal self-check fails. The Hartstone package Workload is commented with guidelines for dealing with several vendors' mathematical library names and function names. By default, it renames the natural logarithm function as "Log," the name proposed by the WG9 Numerics Rapporteur Group [WG9 89].[3]

Pre-Defined Types. The predefined types Integer and Float are used within Hartstone on the assumption that most implementations of these types provide sufficient range and accuracy for Hartstone needs. The counts of met and missed deadlines computed by Hartstone, for example, are expected to be much less than the maximum integer value of a 16-bit machine, and a floating-point type with 6 digits of accuracy provides one-microsecond accuracy for Hartstone timing calculations performed in floating-point. However, before running the Hartstone, the user should check the Digits attribute of the integer and floating-point types to ensure that they meet these range and accuracy assumptions.

[3]The WG9 (Working Group 9) proposal defines the specification of a generic package of elementary functions and a package of related exceptions. Its content derives from a joint proposal of the association for Computing Machinery (ACM) SIGAda Numerics Working Group and the Ada-Europe Numerics Working Group. Draft 1.1 (October 1989) of the proposal has been submitted for consideration as an international standard.

4. Running Hartstone Experiments

The Hartstone benchmark is primarily for embedded real-time target processors that are connected to a host system from which the executable Hartstone code is downloaded. Because of this, and for portability, it is assumed that the only code executing on the target system is the Hartstone code and the Ada runtime system. The Hartstone benchmark makes no explicit calls to Ada runtime system functions or to any kernel operating system layer interposed between it and the Ada runtime system. Additionally, and in particular, no assumptions are made about the Ada runtime system support of host-target file I/O or interactive screen I/O. Therefore, all experiment characteristics (e.g., test duration, task set characteristics, experiment number, experiment completion criterion, etc.) must be known at compile time: in this implementation they cannot be entered interactively or read from a host file. Similarly, the benchmark does not attempt to open any file on the host for output of results. At a minimum, it is expected that the output procedures of the Text_IO package will be capable of writing output to a terminal connected to the target processor. In the SEI host-target environment, the serial ports of the various targets are connected to corresponding serial ports on the VMS host. Output from the targets is displayed in a window on the host console as it arrives at the host serial port. Some cross-compilers provide the capability to capture such host input automatically in a file; for those that do not, the /LOG qualifier of the VMS DCL command SET HOST/DTE/LOG <port_ID> will create a log file of all input arriving at the host serial port.

A user of Hartstone performs one experiment per download. The benchmark is not set up to do multiple experiments per download; the idea is that each separately downloaded experiment begins with the runtime system in the same initial state. To choose an experiment to perform, a user modifies one line in the body of the Experiment package. The criterion for stopping the experiment (for example, stop after a total of 50 deadlines have been missed) may also be set in the next line. By default, the experiment outputs the results of each test in an experiment as the test completes. This is useful for monitoring the progress of an experiment. The user may disable this "full output" option in favor of simply producing a summary of the entire experiment when the experiment completes. Instructions for making these changes are provided as comments in the body of the Experiment package in a section clearly marked as the user-modifiable section. This section also defines two string variables that should be initialized by a user to provide a brief description (e.g., name, version number, target CPU type) of the compiler and target processor. Following these modifications, the package body must then be re-compiled, and the Hartstone benchmark re-linked to produce a new executable module for the chosen experiment.

The default duration of a Hartstone test is 10 seconds, with a 5-second lag before the first test of an experiment begins. If full output is enabled (i.e., if complete test results are to be output as soon as the test completes) and nothing has happened 20 seconds, say, after the start of an experiment, then either Hartstone is broken or there is a host-target communication problem. Of course, if full output is disabled (i.e., no output is produced until the experiment finishes), a user should be prepared to wait a relatively long time to see the summary results.

5. Understanding Hartstone Results

5.1. Format of Results

By default, the Hartstone benchmark outputs the results of every test of an experiment as each test completes. It then prints a summary of the results of the entire experiment. The two-part output from a single test, including the characteristics of a test and its results, is shown below.

```
Experiment:     EXPERIMENT_1
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 1122.19

Test 21 characteristics:

   Task   Frequency    Kilo-Whets   Kilo-Whets   Requested Workload
    No.    (Hertz)     per period   per second      Utilization
     1       2.00          32          64.00          5.70 %
     2       4.00          16          64.00          5.70 %
     3       8.00           8          64.00          5.70 %
     4      16.00           4          64.00          5.70 %
     5     352.00           2         704.00         62.73 %
                                      -------        --------
                                      960.00         85.55 %

Experiment step size:    2.85 %

------------------------------------------------------------------------

Test 21 results:

Test duration (seconds):  10.0

   Task    Period      Met        Missed      Skipped      Average
    No.   in msecs   Deadlines   Deadlines   Deadlines   Late (msec)
     1    500.000        0           7           13        626.683
     2    250.000       40           0            0          0.000
     3    125.000       80           0            0          0.000
     4     62.500      160           0            0          0.000
     5      2.841     3520           0            0          0.000
```

The *raw speed* of the benchmark is the number of Kilo-Whetstone instructions per second (KWIPS) achieved by the Small_Whetstone procedure. This calibration test is performed by the Experiment package when an experiment is initialized. The resultant non-tasking workload rate will always be better than that achievable by splitting the same workload among the five Hartstone tasks; it provides a metric against which the performance of the Hartstone task set can be measured. Both the raw speed calibration test and a Hartstone task include the overhead of calling the Small_Whetstone procedure. The performance requested of Hartstone tasks is expressed as a percentage workload *utilization*, which is computed as the ratio of the requested task speed (in KWIPS) and the raw benchmark speed. The raw speed is assumed to represent 100% utilization. The utilization required of the entire task set is the sum of the individual task utilizations. Successive tests in an experiment increase the requested utilization to the point where deadlines are not met.

The *step size* of an experiment is an indication of the extra work required of the task set when the next test in an experiment is derived from the current test. Like the workload utilization, it is expressed as a percentage of the raw speed. As an example, for experiment 1 the extra work for the task set comes from increasing the frequency of the highest-frequency task, task 5. The additional work required of task 5 is its workload multiplied by the frequency increment defined for task 5 in experiment 1 (in the above example, it is 2 Kilo-Whetstones times 16 Hertz, giving 32 KWIPS). This KWIPS figure, expressed as a percentage of the raw speed KWIPS figure, is the step size for the experiment. It varies from experiment to experiment but remains constant for a specific experiment. The sum of the total requested utilization and the step size for the current test is equal to the next test's total requested utilization. The step size is the granularity, or resolution, of an experiment.

The sum of the met, missed, and skipped deadlines for a task should, in general, be equal to the task's frequency multiplied by the test duration (i.e., the expected number of activations for that task). The case where they do not add up will be discussed later. The average late figure for a task is the average amount by which the task missed its deadlines during the test. It is the sum of the amounts by which individual deadlines were missed, divided by the number of missed deadlines. For lower-priority tasks, it is an indication of the amount of preemption by higher-priority tasks. Skipped deadlines do not influence this figure; they are simply part of the process of "resetting" a task whose lateness is already known.

The summary output produced at the end of an experiment consists of four test results similar to those shown above. The four tests are: the first test in the experiment (the baseline test), the test with the highest utilization and no missed/skipped deadlines (the "best" test), the test which first experienced missed/skipped deadlines, and the final test performed. Example summary results for all four experiments are given in Appendix A.

5.2. The Baseline Test

To get meaningful results from the Hartstone benchmark it is important to define an appropriate starting point for Hartstone experiments. This starting point is the baseline task set and it must first be "tuned" for a user's cross-compiler and target before Hartstone can be used effectively. At a very basic level, "tuning" ensures that the baseline workloads and frequencies are such that an experiment neither runs hundreds of tests before completing, nor completes after running just a few tests. More importantly, a badly-chosen baseline test can lead to unexpected results (discussed later) that undermine the usefulness of the benchmark. This section will provide some guidelines for choosing an appropriate baseline test.

To determine if the characteristics of the baseline task set need to be modified, a user must run a Hartstone experiment "as is" and examine the output of the baseline test. The numbers to check are the total workload utilization and the experiment step size. Every experiment first runs the baseline test, so the total utilization of the baseline test is the same for all experiments. The total utilization should be in the range of 10 percent to 30 percent, so that an experiment commences with a workload rate that is neither too low nor too high (a 50% utilization for the task set in the very first test, for example, would be considered too high). In the example shown in Section 5.1, the total workload utilization of the baseline task set is 28.50 percent (5 times 5.70%). If utilization falls outside the recommended range, the user must edit the task frequencies and/or workloads in the body of package Experiment to bring them into line. If total utilization falls below the range, the task set frequencies and/or workloads must be increased; if it falls above, they must be reduced.

The experiment step size, which represents the resolution of the total utilization, should also be within a range that ensures that the transition from one test to another does not cause either a very large or a very tiny increase in the total resolution. A step size of around 2 or 3 percent seems to be adequate. Step size depends on the parameters controlling the transition from one test to the next. It remains constant for a specific experiment, but varies among different experiments. For experiment 1, it depends on the frequency increment for the highest-frequency task; for experiment 2, it depends on the scale factor applied to all frequencies; for experiment 3, on the workload increment; and for experiment 4, on the frequency and workload of the extra task added for each new test. In the example, the step size is 2.85 percent (task 5's frequency increment times task 5's workload is 16 times 2, which is 32 KWIPS; this is divided by the raw speed, 1122.19 KWIPS, and multiplied by 100 to give 2.85). In general, adjusting the total utilization of the task set will also yield a reasonable step size, so the user should not need to modify the step size parameters.

When making adjustments to the baseline test, the user must be careful to keep the task frequencies harmonic, and must ensure, for example, that the frequency increment of experiment 1 also preserves the harmonic nature of the task set. Workloads must be integral values (the Small_Whetstone benchmark does not permit fractional workloads), so a task cannot be assigned a workload lower than one Kilo-Whetstone per period. By convention, workloads are such that the workload rate (in Kilo-Whetstones per second) of each task in the baseline set is the same.

It is possible for a baseline task set to be within the guidelines just described and yet still fail to run the baseline test successfully. Sections 5.4 and 5.5 provide some answers to this problem.

5.3. What the Results Mean

For any experiment there is no single number which best represents the result of the experiment. The nature of the experiment and the performance of the various Hartstone tasks must be taken into account when formulating a conclusion about the outcome of an experiment. Additionally, the results from all four experiments must be considered when the benchmark is used to evaluate the performance of an Ada runtime system.

The test result of most interest to a user of the Hartstone benchmark is the one representing the highest achieved utilization for an experiment, with no missed or skipped deadlines. In the cases where the experiment is allowed to continue until a predefined number of deadlines have been missed or skipped, the result of the final test run is also of interest because it will show whether or not tasks missed their deadlines in the expected manner for harmonic tasks: the lowest-priority (lowest-frequency) task missing deadlines first, then the next-lowest-priority task, and so on up to the highest-priority (highest-frequency) task.

In each experiment, the step size for that experiment is very significant. The maximum achievable total utilization is represented with a granularity equal to the the experiment step size. Experiments 2 and 3, which affect all 5 tasks, tend to have larger step sizes than experiments 1 and 4, which affect only 1 task.

Once the effect of the step size on the experiment results is understood, the three most important numbers for a test are the total number of task activations, the raw speed, and the total utilization. The total number of activations (equal to the sum of the met plus missed plus skipped deadlines for the task set) is an indication of the amount of task switching overhead required of the runtime system. The total utilization is a measure of the useful work performed, while the raw speed is an upper bound on the amount of useful work capable of being performed.

For experiment 1, the utilization achieved by the highest-frequency task is important since it dominates the overall result (the utilization of the other tasks remains constant throughout the experiment). The maximum frequency achieved by task 5 is of considerable interest since it is the primary indication of the amount of overhead required of the runtime system. As task 5's period decreases, runtime overhead consumes an increasing percentage of the task's period. It is expected that the total utilization for experiment 1 will be lower than that of experiments 2 and 3 because task switching is the predominant factor.

For experiment 2, the utilization of each task is the same for a given test and increases uniformly from one test to the next as all the task frequencies are scaled up. The scaling has the effect of also increasing all task workload rates (as measured in Kilo-Whetstones per second).

For experiment 3, the highest-frequency task's utilization is again of interest because increasing the actual workload, while keeping the frequency constant, means that the workload consumes an increasingly large percentage of this task's period. This, of course, is true for all tasks in this experiment, but the effect is greatest for the highest-frequency task. Experiment 3 should, in general, have better total utilization than the other experiments, since only the workloads increase while the tasks' switching overhead remains the same. A large step size, however, may cause experiment 3's best test result to occur at a lower utilization level: the increase in requested utilization, in the transition from success to missed deadlines, may hide the fact that a smaller increase could have resulted in success at a higher level.

For experiment 4, the utilization of each task remains constant throughout the experiment, but the number of tasks, and hence the total utilization, increases. Of primary interest is the count of extra tasks added to the baseline set. This provides an indication of the runtime system's ability to handle a large number of tasks efficiently.

5.4. Factors Affecting Hartstone Performance

The principal factors affecting the performance of Hartstone PH tests are

- The implementation of task periodicity
- The resolution of the **delay** statement
- The resolution of Calendar.Clock
- The accuracy of the fixed-point type Duration
- The implementation of mathematical library functions
- Floating-point precision
- Miscellaneous overhead factors

Task Periodicity. The implementation of task periodicity in the Hartstone benchmark is based on the paradigm exhibited in Section 9.6 of the Ada Language Reference Manual [LRM 83], a version of which is shown below.

```
declare
  use Calendar;
  -- Period is a global constant of type Duration
  Next_Start : Time := Clock + Period;
begin
  loop
    Next_Delay := Next_Start - Clock;
    delay Next_Delay;
    -- do some work
    Next_Start := Next_Start + Period;
  end loop;
end;
```

This is a highly portable method of implementing periodic tasks in Ada. It is, of course, very dependent on how well the Ada runtime system implements Calendar.Clock and the **delay** statement. At a basic level, the performance of the Hartstone benchmark is a reflection of the performance of these two features of the Ada language. The issues arising from the implementation of these two features are discussed separately below. The other major issue associated with the above paradigm is the possibility of preemption of the task between the reading of the Clock and the start of the **delay** statement, resulting in an actual delay that is longer than the requested delay. It can be shown that this is not a problem for the periodic harmonic task sets used in the Hartstone benchmark.

Delay Statement Resolution. The resolution of the **delay** statement is how closely an actual delay matches a requested delay. A requested delay of one millisecond that is actually implemented as a ten or twenty millisecond delay will cause periodic tasks to start missing deadlines earlier than expected. It has also been implicitly assumed that the expiry of the **delay** statement is preemptive, i.e., that a lower-priority task currently executing will be preempted by a higher-priority task whose delay has expired. A non-preemptive **delay** statement will likely cause results that are at least as poor as, and probably worse than, those for a coarse **delay** statement resolution. Implementations using non-preemptive delays are technically non-conforming, but the current Ada Compiler Validation Capability (ACVC, Version 1.10) does not adequately test this.

Calendar.Clock Resolution. The resolution of Calendar.Clock is the time period between successive ticks of the clock. A Hartstone task performs arithmetic involving Calendar.Clock to determine the time remaining in its period upon completion of its workload. It then suspends itself by delaying until its computed "wakeup" time—the next scheduled activation time. A coarse Calendar.Clock resolution means that a coarse value will be used as the expression in the **delay** statement, thereby resulting in a flawed implementation of task periodicity. Also, a coarse clock resolution may cause variations in the calibrated raw speed of the Small_Whetstone procedure. There are large differences in the resolution of Calendar.Clock in current Ada cross-compilers, ranging, in those tested at the SEI, from 61 microseconds to 100 milliseconds. The ACM SIGAda Performance Issues Working Group (PIWG)[4] benchmark suite contains tests to measure the resolution of Calendar.Clock and the **delay** statement. These resolutions should always be checked by users of Hartstone. (Note that, in general, the value of System.Tick is not the same as the resolution of Calendar.Clock; a test should always be performed to determine the actual resolution.) Sample results of these two tests, for the XD Ada MC68020 cross-compiler, are included in Appendix B.

Type Duration. The accuracy of type Duration can be determined by examining the value of Duration'Small. For many implementations, this value is 2^{-14} seconds, or approximately 61 microseconds. For some implementations, however, the value is 1 millisecond. In an attempt to minimize the cumulative errors possible in fixed-point Duration arithmetic, a Hartstone periodic task actually performs all arithmetic involving the types Time and Duration in floating-point. This is done by using floating-point variables to compute Next_Start and Next_Delay and converting Next_Delay to type Duration in the actual **delay** statement. The value returned by Calendar.Clock is of the private type Time and so cannot be converted directly. Instead the Calendar.Seconds function is used to extract the seconds portion of the Time value; this value is of the non-private type Day_Duration and so is amenable to direct conversion.[5]

Mathematical Library. The raw non-tasking speed measurement of the Small_Whetstone procedure is another important factor since it is the basis for the utilization figures and the experiment step size. The raw speed will depend on how efficiently the Small_Whetstone computations are performed. For example, the computations involve trigonometric, logarithmic, and exponential functions whose efficiency depends on whether they are implemented wholly in software on the main processor, or by special instructions on a co-processor, if one is present on the target board. Testing at SEI has shown that most mathematical libraries do take advantage of an on-board co-processor, but that even when they do, the differences in the performance of Hartstone's Small_Whetstone (and the PIWG full Whetstone benchmark) on the same target board are striking.

[4]The name, address, and telephone number of the current PIWG chairperson and other officers may be found in Ada Letters, a bimonthly publication of the ACM Special Interest Group on Ada (SIGAda)

[5]Because the seconds portion of the time value becomes zero after twenty-four hours, you should not run Hartstone through a midnight boundary. Depending on how **Calendar.Clock** is initialized, "midnight" for the target system may bear no relation to midnight as measured by a wall clock (which in turn may be different from time as measured by the host system).

Floating-Point Precision. The current implementation of Hartstone uses the type Float for all floating-point computations. Of the 8 Ada cross-compilers at the SEI, 7 implement type Float with 6 decimal digits of precision (Float'Digits = 6) while 1 implements it with 15. Rather than defining a machine-dependent package that simply contains a type Hart_Float, say, Hartstone uses the type Float on the assumption that it will always provide at least 6 digits of precision. Doing the computational workload of Hartstone (the Small_Whetstone procedure) in a higher-precision floating-point type may, of course, take longer. The user must be aware of this when comparing Hartstone results from different Ada implementations. For consistency, a floating-point type with 6 digits of precision should be used; this will usually be the predefined type Float, but, for some cross-compilers, may be Short_Float.

Miscellaneous Overhead Factors. Calling the Small_Whetstone procedure from within a Hartstone task is another factor affecting performance; the overhead of the call may be zero if in-lining is used and non-zero otherwise. Again, the PIWG suite provides tests to measure this overhead. Hartstone contains an inline **pragma** for Small_Whetstone; the user should check the compilation listings to see if the compiler is accepting or rejecting it. Even when the pragma is accepted there may still be a performance factor attributable to the location and the even/odd word alignment of the copies of the code in different areas of memory.

There are other sources of overhead which undoubtedly influence Hartstone but are difficult for users to measure. These include, but are not limited to, the tasks' switching time, time spent in the clock interrupt handler, time spent managing delay and ready queues upon expiry of a delay, cache hit/miss rates, time to switch between the processor and co-processor, and, possibly, periodic garbage collection. Highly-specific, fine-grained benchmark tests, or hardware timing capabilities such as those provided by a logic analyzer, are needed to detect and measure the effect of such items on Hartstone's performance.

5.5. Unexpected Results

In normal circumstances, a Hartstone experiment proceeds from the baseline test through a number of intermediate tests to a point where a test meets the predefined completion criterion for the experiment. The results of the experiment can then be examined to determine the overall utilization and the failure pattern when tasks began to miss their deadlines. Sometimes the results can be quite different from what the user expected. This section attempts to characterize a sample set of such results; it is based on actual results encountered during testing of Hartstone on various Ada cross-compilers and target processors.

Baseline Test Failure. As discussed earlier, one reason for this may be the fact that the baseline task set utilization is outside the recommended range. However, even when it is within range, other factors may cause missed deadlines in the baseline set. A non-preemptive **delay** statement, or one with poor resolution, means that the actual implemented frequency of a task is much less than the requested frequency. Since a task's period and activation times are computed as a function of the *requested* frequency, an implemented frequency that is lower will cause a task to delay needlessly and miss its scheduled activation times. Even a reasonable **delay** statement resolution can still be overwhelmed when used in combination with a Calendar.Clock with poor resolution to implement task periodicity. The user's only recourse is to scale back the frequencies of the baseline task set (keeping them harmonic) and re-run the experiment. A rule of thumb: the benchmark is already in trouble if the period of the highest-frequency baseline task is less than the period between successive ticks of Calendar.Clock. For example, if the highest-frequency baseline task's frequency is 32 Hertz and the resolution of Calendar.Clock is 100 milliseconds, the task's requested 31.25-millisecond period will never be realized. The outcome may well be that Hartstone cannot manage a successful run of even the first test without scaling back the baseline task set. One possible, but highly machine-dependent solution to the problem is to use a high-resolution programmable timer (if one is available on the target system) as a source of periodic interrupts. A dispatcher program could field these interrupts and dispatch tasks at their assigned frequencies in the manner described in [Borger 89].

Excess Task Activations. When a periodic task runs at a fixed frequency, measured in task activations per unit time, in a test whose duration is a multiple of the unit time, then the number of times the task can be expected to activate is the product of the task frequency and the test duration. In the Hartstone benchmark, the outcome of any one run of a Hartstone periodic task will be a met, missed, or skipped deadline; therefore the sum of all such met, missed, and skipped deadlines reported by the task in a single test will equal the actual count of activations for that task. Testing has shown that, for the highest-frequency task of experiment 1, the actual activation count sometimes exceeds the expected activation count. The reason has to do with the way periodic tasks, in this implementation, keep track of time. A task starts at its assigned starting time, performs its assigned workload, and determines its next activation time by adding its period to the starting time. Each time around the task's main loop, the new activation time is compared with the test's finishing time (pre-computed by adding the test duration to the starting time) and the task executes for another cycle if the finishing time has not been reached. If the successive additions of the task's period to the starting time eventually yield a value exactly equal to the finishing time then the test finishes without extra activations. Because of rounding effects, however, the task may complete its "expected" number of activations and still manage one or more runs before the finishing time occurs. It is also possible that a coarse Calendar.Clock resolution will allow extra activations; since there is no external timing source in this version of Hartstone (e.g., periodic interrupts from a programmable interval timer, a highly implementation-dependent, non-portable solution), there is no way to cut tasks off at exactly the end of a test.

Inverted Task Set Breakdown Pattern. Because of the priority structure of the task set (highest-frequency task has highest priority, lowest-frequency task has lowest) one expects the lower-frequency tasks to be preempted by the higher-frequency tasks. Thus the expected breakdown pattern for the task set is that task 1 (lowest priority) will miss deadlines first, then task 2, and so on. Tests have shown that this is not always the case. In experiment 1, the frequency of the highest-frequency task is incremented for each new test, with the result that the task-switching overhead becomes an increasingly significant percentage of the task's period. Eventually, the rapid switching required of the task leaves no time for useful work, and the highest-frequency task starts missing deadlines before any of the other tasks start missing theirs. The effect of this breakdown pattern is that the total workload utilization for the task set may be poor, despite the fact that the highest-frequency task may have been driven to a very high frequency before it started to miss deadlines. Tests have shown that the inverted breakdown pattern usually occurs if the total utilization of the baseline task set is less than 10 percent. The user should scale up the baseline characteristics (remembering to keep the task set frequencies harmonic) to overcome the problem.

Inverted Summary Results. During testing of Hartstone, the highest-frequency task of experiment 1 would sometimes miss a single deadline, but then meet all its deadlines in the next several tests. The experiment would continue normally until the task set began missing deadlines in the expected fashion, at which point the experiment would terminate. This situation can be detected by examining the summary reports produced at the end of an experiment. One of the summaries is the output of the "best" test—the one achieving the highest utilization with no missed deadlines. Another summary is the output of the test where deadlines were first missed. The test number of the "best" test normally precedes that of the "first missed" test; however, in the case where a test with missed deadlines is followed by one or more tests that do not miss deadlines, the test number of the "best" test is consequently higher than that of the "first missed" test. This phenomenon is still under investigation; preliminary testing with a logic analyzer indicates that the highest-frequency task may be blocked for varying amounts of time by runtime system activities such as delay queue management and Calendar.Clock updating. Depending on the amount of queue re-organization required, and whether or not the clock also needs servicing, the highest-frequency task may occasionally be blocked just long enough to miss a deadline.

Exceptions. The Small_Whetstone procedure raises an exception if it fails an internal check on the result of its computation. Two reasons for such a failure have been encountered during testing. The first was when the link-time memory layout parameters did not allow enough stack and heap space in the target board's memory for Hartstone. A simple readjustment of the parameters took care of the problem. The second reason was more subtle, involving different interpretations of the name "Log" as used in vendor mathematical libraries to denote a logarithm function. The logarithm function used within the Small_Whetstone procedure is intended to be the natural logarithm function (base e), not the base 10 function. Some vendors denote the former by "Ln" and the latter by "Log"; others use "Log" for natural logarithms and a name such as "Log10" for base 10 logarithms. If base 10 logs are used inadvertently (i.e., the user did not modify the Small_Whetstone procedure correctly for the mathematical library being used) the compilation will succeed but the computation performed by Small_Whetstone will produce a runtime exception.

Other exceptions, such as Storage_Error, can arise if not enough code space has been allocated for Hartstone (again, modifying the file that describes the target memory layout solves the problem), or if the runtime system provides support only for a default number of tasks (possibly defined by a user-modifiable link parameter) and this default is exceeded by the extra tasks created in experiment 4.

6. Future Work

It is expected that this report will be sufficient to enable a Hartstone user to run a series of experiments against a particular Ada compiler on a particular architecture. The sample outputs show what experiment results look like and some initial guidance on interpretation of results has been provided. However, in order to be a truly useful tool, it is necessary to be able to compare different implementations and provide a deeper analysis of results. Work is under way at the SEI to do just that. The Hartstone benchmark will be used to generate results for several different embedded systems cross-compilers. A subsequent report will describe these results and the analysis required to draw from them conclusions about the usability of the cross-compilers for hard real-time applications. The purpose of the report will not be to "rate" the various cross-compilers, but to show Hartstone users how to draw their own conclusions when evaluating the hard real-time characteristics of their own Ada compilers.

Bibliography

[Borger 89]	Borger, M., Klein, M., Veltre, R. *Real-Time Software Engineering in Ada: Observations and Guidelines.* Technical Report CMU/SEI-89-TR-22, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213, September, 1989.
[Curnow 76]	Curnow, H.J. and Wichmann, B.A. A Synthetic Benchmark. *Computer Journal* 19(1):43-49, January, 1976.
[Harbaugh 84]	Harbaugh, S. and Forakis, J. Timing Studies using a Synthetic Whetstone Benchmark. *Ada Letters* 4(2):23-34, 1984.
[Liu 73]	Liu, C.L. and Layland, J.W. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the Association of Computing Machinery* 20(1):46-61, January, 1973.
[LRM 83]	United States Department of Defense. *Reference Manual for the Ada Programming Language* American National Standards Institute, New York, 1983.
[Sha 89]	Sha, L. and Goodenough, J.B. *Real-Time Scheduling Theory and Ada.* Technical Report CMU/SEI-89-TR-14, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213, April, 1989.
[Weiderman 89]	Weiderman, Nelson. *Hartstone: Synthetic Benchmark Requirements for Hard Real-Time Applications.* Technical Report CMU/SEI-89-TR-23, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213, June, 1989.
[WG9 89]	ISO-IEC/JTC1/SC22/WG9 (Ada) Numerics Rapporteur Group. *Proposed Standard for a Generic Package of Elementary Functions for Ada* WG9 Numerics Rapporteur Group, 1989.
[Wichmann 88]	Wichmann, B.A. *Validation Code for the Whetstone Benchmark.* Technical Report DITC 107/88, National Physical Laboratory, Teddington, Middlesex, UK, March, 1988.

Appendix A: Sample Results for XD Ada VAX/VMS -> MC68020

A.1. Host-Target Configuration

The following is the host-target configuration used for generating the results reported here:

HOST:	DEC MicroVAX II running VAX/VMS, Release 5.1-1
CROSS-COMPILER:	Systems Designers XD Ada, Version 1.0, ACVC 1.10
TARGET:	Motorola MVME133: 12.5 MHz MC68020 CPU with 12.5 MHz MC68881 Floating-Point Co-processor; one wait state; 1Mb RAM; 256-byte on-chip instruction cache

Full optimization (the default) was specified for all compilations. No checks were suppressed. The summary output for the four Hartstone experiments is shown in the next four sections.

A.2. Experiment 1: Summary of Results

```
                    HARTSTONE BENCHMARK SUMMARY RESULTS

Baseline test:

==========================================================================

Experiment:     EXPERIMENT_1
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 1122.19

Test 1 characteristics:

   Task   Frequency   Kilo-Whets   Kilo-Whets   Requested Workload
    No.    (Hertz)    per period   per second      Utilization
    1        2.00         32          64.00           5.70 %
    2        4.00         16          64.00           5.70 %
    3        8.00          8          64.00           5.70 %
    4       16.00          4          64.00           5.70 %
    5       32.00          2          64.00           5.70 %
                                     -------        --------
                                      320.00         28.52 %

Experiment step size:    2.85 %

--------------------------------------------------------------------------

Test 1 results:

Test duration (seconds):  10.0

   Task    Period      Met       Missed     Skipped      Average
    No.   in msecs  Deadlines  Deadlines  Deadlines  Late (msec)
    1     500.000      20          0          0          0.000
    2     250.000      40          0          0          0.000
    3     125.000      80          0          0          0.000
    4      62.500     160          0          0          0.000
    5      31.250     320          0          0          0.000

==========================================================================
Last test with no missed/skipped deadlines:

========================================================================

Experiment:    EXPERIMENT_1
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 1122.19

Test 20 characteristics:

   Task   Frequency   Kilo-Whets   Kilo-Whets   Requested Workload
    No.    (Hertz)    per period   per second      Utilization
     1       2.00         32          64.00           5.70 %
     2       4.00         16          64.00           5.70 %
     3       8.00          8          64.00           5.70 %
     4      16.00          4          64.00           5.70 %
     5     336.00          2         672.00          59.88 %
                                     -------         --------
                                     928.00          82.70 %

Experiment step size:    2.85 %

------------------------------------------------------------------------

Test 20 results:

Test duration (seconds):  10.0

   Task     Period      Met        Missed      Skipped       Average
    No.   in msecs   Deadlines   Deadlines   Deadlines   Late (msec)
     1    500.000       20           0           0           0.000
     2    250.000       40           0           0           0.000
     3    125.000       80           0           0           0.000
     4     62.500      160           0           0           0.000
     5      2.976     3360           0           0           0.000

========================================================================
Test when deadlines first missed/skipped:

=======================================================================

Experiment:    EXPERIMENT_1
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 1122.19

Test 21 characteristics:

   Task   Frequency   Kilo-Whets   Kilo-Whets   Requested Workload
    No.    (Hertz)    per period   per second      Utilization
     1       2.00         32          64.00           5.70 %
     2       4.00         16          64.00           5.70 %
     3       8.00          8          64.00           5.70 %
     4      16.00          4          64.00           5.70 %
     5     352.00          2         704.00          62.73 %
                                     -------        --------
                                      960.00          85.55 %

Experiment step size:    2.85 %

-----------------------------------------------------------------------

Test 21 results:

Test duration (seconds):  10.0

   Task    Period      Met       Missed     Skipped     Average
    No.   in msecs  Deadlines  Deadlines  Deadlines  Late (msec)
     1    500.000       0          7          13       626.683
     2    250.000      40          0           0         0.000
     3    125.000      80          0           0         0.000
     4     62.500     160          0           0         0.000
     5      2.841    3520          0           0         0.000

=======================================================================
```

Final test performed:

```
==========================================================================

Experiment:    EXPERIMENT_1
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 1122.19

Test 22 characteristics:

   Task  Frequency   Kilo-Whets   Kilo-Whets   Requested Workload
    No.   (Hertz)    per period   per second      Utilization
    1       2.00         32          64.00           5.70 %
    2       4.00         16          64.00           5.70 %
    3       8.00          8          64.00           5.70 %
    4      16.00          4          64.00           5.70 %
    5     368.00          2         736.00          65.59 %
                                   -------        --------
                                    992.00          88.40 %

Experiment step size:   2.85 %

--------------------------------------------------------------------------

Test 22 results:

Test duration (seconds):  10.0

   Task   Period      Met      Missed     Skipped      Average
    No.  in msecs  Deadlines  Deadlines  Deadlines   Late (msec)
    1    500.000       0          6          14       1095.724
    2    250.000       0         20          20        103.137
    3    125.000      80          0           0          0.000
    4     62.500     160          0           0          0.000
    5      2.717    3680          0           0          0.000

==========================================================================
Benchmark : Hartstone Benchmark, Version 1.0
Compiler  : Systems Designers XD Ada 1.0 VAX/VMS -> MC68020
Target    : Motorola MVME133 (12.5 MHz MC68020 & 12.5 MHz MC68881)

Characteristics of best test for this experiment:
(no missed/skipped deadlines)

   Test 20 of Experiment 1

   Raw (non-tasking) benchmark speed in KWIPS: 1122.19

   Full task set:

      Total      Deadlines      Task Set       Total
      Tasks      Per Second    Utilization     KWIPS
        5          366.00        82.70 %       928.00

   Highest-frequency task:

      Period     Deadlines        Task         Task
      (msec)     Per Second    Utilization     KWIPS
       2.976       336.00        59.88 %       672.00

   Experiment step size:   2.85 %

               END OF HARTSTONE BENCHMARK SUMMARY RESULTS
```

A.3. Experiment 2: Summary of Results

```
                    HARTSTONE BENCHMARK SUMMARY RESULTS

Baseline test:

==========================================================================

Experiment:    EXPERIMENT_2
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 1122.10

Test 1 characteristics:

   Task  Frequency   Kilo-Whets   Kilo-Whets   Requested Workload
    No.   (Hertz)    per period   per second      Utilization
    1      2.00          32         64.00            5.70 %
    2      4.00          16         64.00            5.70 %
    3      8.00           8         64.00            5.70 %
    4     16.00           4         64.00            5.70 %
    5     32.00           2         64.00            5.70 %
                                   -------         --------
                                    320.00          28.52 %

Experiment step size:   2.85 %

--------------------------------------------------------------------------

Test 1 results:

Test duration (seconds):  10.0

   Task   Period     Met        Missed      Skipped      Average
    No.  in msecs  Deadlines  Deadlines   Deadlines   Late (msec)
    1    500.000      20          0           0          0.000
    2    250.000      40          0           0          0.000
    3    125.000      80          0           0          0.000
    4     62.500     160          0           0          0.000
    5     31.250     320          0           0          0.000

==========================================================================
Last test with no missed/skipped deadlines:

===========================================================================

Experiment:    EXPERIMENT_2
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 1122.10

Test 23 characteristics:

   Task   Frequency   Kilo-Whets   Kilo-Whets   Requested Workload
    No.    (Hertz)    per period   per second      Utilization
     1       6.40         32         204.80          18.25 %
     2      12.80         16         204.80          18.25 %
     3      25.60          8         204.80          18.25 %
     4      51.20          4         204.80          18.25 %
     5     102.40          2         204.80          18.25 %
                                     -------         --------
                                     1024.00         91.26 %

Experiment step size:    2.85 %

---------------------------------------------------------------------------

Test 23 results:

Test duration (seconds):  10.0

   Task    Period      Met       Missed      Skipped      Average
    No.   in msecs  Deadlines  Deadlines   Deadlines   Late (msec)
     1    156.250      64          0           0          0.000
     2     78.125     128          0           0          0.000
     3     39.063     256          0           0          0.000
     4     19.531     512          0           0          0.000
     5      9.766    1024          0           0          0.000

===========================================================================
Test when deadlines first missed/skipped:

==============================================================

Experiment:     EXPERIMENT_2
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 1122.10

Test 24 characteristics:

   Task  Frequency  Kilo-Whets  Kilo-Whets  Requested Workload
    No.   (Hertz)   per period  per second     Utilization
     1      6.60        32        211.20        18.82 %
     2     13.20        16        211.20        18.82 %
     3     26.40         8        211.20        18.82 %
     4     52.80         4        211.20        18.82 %
     5    105.60         2        211.20        18.82 %
                                 -------       --------
                                 1056.00        94.11 %

Experiment step size:   2.85 %

--------------------------------------------------------------

Test 24 results:

Test duration (seconds):  10.0

   Task   Period      Met       Missed    Skipped     Average
    No.  in msecs  Deadlines  Deadlines  Deadlines  Late (msec)
     1   151.515       0         33         33        55.840
     2    75.758     132          0          0         0.000
     3    37.879     264          0          0         0.000
     4    18.939     528          0          0         0.000
     5     9.470    1056          0          0         0.000

==============================================================
Final test performed:
   See preceding summary of test 24


=========================================================================

Benchmark : Hartstone Benchmark, Version 1.0
Compiler  : Systems Designers XD Ada 1.0 VAX/VMS -> MC68020
Target    : Motorola MVME133 (12.5 MHz MC68020 & 12.5 MHz MC68881)

Characteristics of best test for this experiment:
(no missed/skipped deadlines)

   Test 23 of Experiment 2

   Raw (non-tasking) benchmark speed in KWIPS: 1122.10

   Full task set:

      Total      Deadlines      Task Set      Total
      Tasks      Per Second    Utilization    KWIPS
        5          198.40        91.26 %     1024.00

   Highest-frequency task:

      Period     Deadlines       Task         Task
      (msec)     Per Second    Utilization    KWIPS
       9.766       102.40        18.25 %     204.80

   Experiment step size:    2.85 %

=========================================================================

               END OF HARTSTONE BENCHMARK SUMMARY RESULTS
```

A.4. Experiment 3: Summary of Results

```
                    HARTSTONE BENCHMARK SUMMARY RESULTS

Baseline test:

========================================================================

Experiment:    EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 1121.88

Test 1 characteristics:

   Task   Frequency   Kilo-Whets   Kilo-Whets   Requested Workload
    No.    (Hertz)    per period   per second       Utilization
     1       2.00         32         64.00            5.70 %
     2       4.00         16         64.00            5.70 %
     3       8.00          8         64.00            5.70 %
     4      16.00          4         64.00            5.70 %
     5      32.00          2         64.00            5.70 %
                                    -------          --------
                                    320.00           28.52 %

Experiment step size:   5.53 %

------------------------------------------------------------------------

Test 1 results:

Test duration (seconds):  10.0

   Task    Period      Met       Missed     Skipped     Average
    No.   in msecs  Deadlines  Deadlines  Deadlines  Late (msec)
     1    500.000      20          0          0          0.000
     2    250.000      40          0          0          0.000
     3    125.000      80          0          0          0.000
     4     62.500     160          0          0          0.000
     5     31.250     320          0          0          0.000

========================================================================
Last test with no missed/skipped deadlines:

==========================================================================

Experiment:    EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 1121.88

Test 13 characteristics:

   Task  Frequency   Kilo-Whets  Kilo-Whets   Requested Workload
    No.   (Hertz)    per period  per second       Utilization
     1      2.00         44        88.00           7.84 %
     2      4.00         28       112.00           9.98 %
     3      8.00         20       160.00          14.26 %
     4     16.00         16       256.00          22.82 %
     5     32.00         14       448.00          39.93 %
                                 -------        --------
                                 1064.00          94.84 %

Experiment step size:   5.53 %

--------------------------------------------------------------------------

Test 13 results:

Test duration (seconds):  10.0

   Task   Period      Met      Missed     Skipped     Average
    No.  in msecs  Deadlines  Deadlines  Deadlines  Late (msec)
     1   500.000      20          0          0          0.000
     2   250.000      40          0          0          0.000
     3   125.000      80          0          0          0.000
     4    62.500     160          0          0          0.000
     5    31.250     320          0          0          0.000

==========================================================================
Test when deadlines first missed/skipped:

===========================================================================

Experiment:    EXPERIMENT_3
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 1121.88

Test 14 characteristics:

   Task  Frequency  Kilo-Whets  Kilo-Whets  Requested Workload
   No.    (Hertz)   per period  per second     Utilization
    1      2.00         45        90.00          8.02 %
    2      4.00         29       116.00         10.34 %
    3      8.00         21       168.00         14.97 %
    4     16.00         17       272.00         24.24 %
    5     32.00         15       480.00         42.79 %
                                 -------        --------
                                 1126.00        100.37 %

Experiment step size:    5.53 %

---------------------------------------------------------------------------

Test 14 results:

Test duration (seconds):  10.0

   Task   Period      Met       Missed     Skipped      Average
   No.   in msecs  Deadlines  Deadlines   Deadlines   Late (msec)
    1    500.000       0         10          10         248.639
    2    250.000      40          0           0           0.000
    3    125.000      80          0           0           0.000
    4     62.500     160          0           0           0.000
    5     31.250     320          0           0           0.000

===========================================================================
Final test performed:
   See preceding summary of test 14


==========================================================================

Benchmark : Hartstone Benchmark, Version 1.0
Compiler  : Systems Designers XD Ada 1.0 VAX/VMS -> MC68020
Target    : Motorola MVME133 (12.5 MHz MC68020 & 12.5 MHz MC68881)

Characteristics of best test for this experiment:
(no missed/skipped deadlines)

   Test 13 of Experiment 3

   Raw (non-tasking) benchmark speed in KWIPS: 1121.88

   Full task set:

      Total      Deadlines       Task Set       Total
      Tasks      Per Second     Utilization     KWIPS
        5          62.00          94.84 %      1064.00

   Highest-frequency task:

      Period     Deadlines        Task          Task
      (msec)     Per Second     Utilization     KWIPS
      31.250       32.00          39.93 %       448.00

   Experiment step size:    5.53 %

==========================================================================


               END OF HARTSTONE BENCHMARK SUMMARY RESULTS
```

A.5. Experiment 4: Summary of Results

In the summaries that follow, the characteristics (frequencies, workloads, and utilizations) of the extra tasks added to the baseline set are all identical; therefore, some have been edited out for brevity. Similarly, some of the identical results produced by these extra tasks have also been omitted. Such omissions are indicated by ellipses.

```
                    HARTSTONE BENCHMARK SUMMARY RESULTS

Baseline test:

===========================================================================

Experiment:     EXPERIMENT_4
Completion on:  Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 1122.11

Test 1 characteristics:

   Task   Frequency   Kilo-Whets   Kilo-Whets   Requested Workload
   No.     (Hertz)    per period   per second      Utilization
    1        2.00         32          64.00             5.70 %
    2        4.00         16          64.00             5.70 %
    3        8.00          8          64.00             5.70 %
    4       16.00          4          64.00             5.70 %
    5       32.00          2          64.00             5.70 %
                                     -------          --------
                                      320.00           28.52 %

Experiment step size:    5.70 %

---------------------------------------------------------------------------

Test 1 results:

Test duration (seconds):  10.0

   Task    Period       Met       Missed      Skipped      Average
   No.    in msecs   Deadlines   Deadlines   Deadlines   Late (msec)
    1     500.000       20          0            0          0.000
    2     250.000       40          0            0          0.000
    3     125.000       80          0            0          0.000
    4      62.500      160          0            0          0.000
    5      31.250      320          0            0          0.000

===========================================================================
Last test with no missed/skipped deadlines:
==========================================================================

Experiment:    EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 1122.11

Test 12 characteristics:

   Task   Frequency   Kilo-Whets   Kilo-Whets   Requested Workload
    No.    (Hertz)    per period   per second      Utilization
    1       2.00         32          64.00            5.70 %
    2       4.00         16          64.00            5.70 %
    3       8.00          8          64.00            5.70 %
    4      16.00          4          64.00            5.70 %
    5      32.00          2          64.00            5.70 %
    6       8.00          8          64.00            5.70 %
    .        .            .            .                .
    .        .            .            .                .
    .        .            .            .                .

   16       8.00          8          64.00            5.70 %
                                    -------          --------
                                    1024.00          91.26 %

Experiment step size:   5.70 %

--------------------------------------------------------------------------

Test 12 results:

Test duration (seconds):  10.0

   Task    Period      Met       Missed     Skipped      Average
    No.   in msecs  Deadlines  Deadlines  Deadlines   Late (msec)
    1     500.000      20          0          0          0.000
    2     250.000      40          0          0          0.000
    3     125.000      80          0          0          0.000
    4      62.500     160          0          0          0.000
    5      31.250     320          0          0          0.000
    6     125.000      80          0          0          0.000
    .        .          .          .          .            .
    .        .          .          .          .            .
    .        .          .          .          .            .

   16     125.000      80          0          0          0.000
==========================================================================
```

Test when deadlines first missed/skipped:

```
=====================================================================

Experiment:     EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 1122.11

Test 13 characteristics:

   Task   Frequency   Kilo-Whets   Kilo-Whets   Requested Workload
    No.    (Hertz)    per period   per second       Utilization
     1       2.00         32         64.00           5.70 %
     2       4.00         16         64.00           5.70 %
     3       8.00          8         64.00           5.70 %
     4      16.00          4         64.00           5.70 %
     5      32.00          2         64.00           5.70 %
     6       8.00          8         64.00           5.70 %
     .        .            .           .               .
     .        .            .           .               .
     .        .            .           .               .

    17       8.00          8         64.00           5.70 %
                                    -------         --------
                                    1088.00         96.96 %

Experiment step size:    5.70 %

---------------------------------------------------------------------

Test 13 results:

Test duration (seconds):  10.0

   Task    Period      Met      Missed     Skipped     Average
    No.   in msecs  Deadlines  Deadlines  Deadlines  Late (msec)
     1    500.000       0         10         10        247.742
     2    250.000      40          0          0          0.000
     3    125.000      80          0          0          0.000
     4     62.500     160          0          0          0.000
     5     31.250     320          0          0          0.000
     6    125.000      80          0          0          0.000
     .       .          .          .          .            .
     .       .          .          .          .            .
     .       .          .          .          .            .

    17    125.000      80          0          0          0.000

=====================================================================
Final test performed:

===========================================================================

Experiment:    EXPERIMENT_4
Completion on: Miss/skip 50 deadlines

Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): 1122.11

Test 14 characteristics:

   Task   Frequency   Kilo-Whets   Kilo-Whets   Requested Workload
    No.    (Hertz)    per period   per second      Utilization
     1       2.00         32         64.00            5.70 %
     2       4.00         16         64.00            5.70 %
     3       8.00          8         64.00            5.70 %
     4      16.00          4         64.00            5.70 %
     5      32.00          2         64.00            5.70 %
     6       8.00          8         64.00            5.70 %
     .         .           .           .                .
     .         .           .           .                .
     .         .           .           .                .

    18       8.00          8         64.00            5.70 %
                                   -------         --------
                                   1152.00          102.66 %

Experiment step size:    5.70 %

---------------------------------------------------------------------------

Test 14 results:

Test duration (seconds):  10.0

   Task    Period       Met       Missed      Skipped       Average
    No.   in msecs   Deadlines   Deadlines   Deadlines   Late (msec)
     1    500.000        0           4          16         2002.884
     2    250.000        0          20          20          124.420
     3    125.000       80           0           0            0.000
     4     62.500      160           0           0            0.000
     5     31.250      320           0           0            0.000
     6    125.000       80           0           0            0.000
     .       .           .           .           .             .
     .       .           .           .           .             .
     .       .           .           .           .             .

    18    125.000       80           0           0            0.000

===========================================================================
================================================================================

Benchmark : Hartstone Benchmark, Version 1.0
Compiler  : Systems Designers XD Ada 1.0 VAX/VMS -> MC68020
Target    : Motorola MVME133 (12.5 MHz MC68020 & 12.5 MHz MC68881)

Characteristics of best test for this experiment:
(no missed/skipped deadlines)

   Test 12 of Experiment 4

   Raw (non-tasking) benchmark speed in KWIPS: 1122.11

   Full task set:

      Total      Deadlines      Task Set      Total
      Tasks      Per Second    Utilization    KWIPS
        16         150.00        91.26 %     1024.00

   Highest-frequency task:

      Period     Deadlines       Task         Task
      (msec)     Per Second    Utilization    KWIPS
      31.250       32.00          5.70 %       64.00

   Experiment step size:    5.70 %

================================================================================
```

END OF HARTSTONE BENCHMARK SUMMARY RESULTS

Appendix B: Supporting PIWG Results

The following are the results of some ACM Performance Issues Working Group (PIWG) benchmarks for XD Ada 1.0 on the Motorola MVME133 board. The tests come from the December 12, 1987 release of the benchmarks. All compilations had full optimization in effect and no checks were suppressed.

B.1. Calendar.Clock Resolution

```
Test Name:   A000090
 Clock resolution measurement running
Test Description:
 Determine clock resolution using second differences
 of values returned by the function CPU_Time_Clock.

 Number of sample values is       12000
 Clock Resolution            =        0.000122070312500 seconds.
 Clock Resolution (average)  =        0.000122070312500 seconds.
 Clock Resolution (variance) =        0.000000000000000 seconds.
```

B.2. Delay Statement Resolution

The delay values shown are in seconds.

```
  Y000001 Measure actual delay vs commanded delay
     Commanded       Actual         CPU     Iterations
      0.0010         0.0013         0.0013   4096
      0.0020         0.0023         0.0023   2048
      0.0039         0.0042         0.0042   1024
      0.0078         0.0081         0.0081   512
      0.0156         0.0159         0.0159   256
      0.0313         0.0314         0.0314   128
      0.0625         0.0626         0.0626   64
      0.1250         0.1252         0.1252   32
      0.2500         0.2501         0.2501   16
      0.5000         0.5000         0.5001   8
      1.0000         1.0001         1.0001   4
      2.0000         2.0002         2.0002   2
      4.0000         4.0001         4.0002   2
      8.0000         8.0001         8.0002   2
```

B.3. Procedure Call Overhead

```
Test Name:    P000005                          Class Name:  Procedure
CPU Time:         1.6  microseconds
Wall Time:        1.6  microseconds.           Iteration Count:   1024
Test Description:
 Procedure call and return time measurement
 The procedure is in a separately compiled package
 One parameter, in INTEGER
```

```
Test Name:    P000006                          Class Name:  Procedure
CPU Time:         2.8  microseconds
Wall Time:        2.8  microseconds.           Iteration Count:   1024
Test Description:
 Procedure call and return time measurement
 The procedure is in a separately compiled package
 One parameter, out INTEGER
```

```
Test Name:    P000007                          Class Name:  Procedure
CPU Time:         3.1  microseconds
Wall Time:        3.1  microseconds.           Iteration Count:   1024
Test Description:
 Procedure call and return time measurement
 The procedure is in a separately compiled package
 One parameter, in out INTEGER
```

Appendix C: Obtaining Hartstone Source Code and Information

Hartstone source code and supporting documentation can be obtained from the Real-Time Embedded Systems Testbed (REST) Project at the Software Engineering Institute in a number of different ways. Full details can be obtained by sending a request for information to the electronic mail or postal address listed below.

Electronic mail requests should be sent to the following Internet address:

HARTSTONE-INFO@SEI.CMU.EDU

Electronic mail received at this address will automatically return to the sender instructions on all available distribution mechanisms.

For people who do not have Internet access, the address to send information requests to is:

REST Transition Services
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213-3890
Phone: (412) 268-7700

Appendix D: Hartstone Ada Code for PH Series

The code in this appendix is listed in the order shown below.

Main procedure:	Hartstone
Package spec:	Experiment
Package body:	Experiment
Package spec:	Periodic_Tasks
Package body:	Periodic_Tasks
Package spec:	Workload
Package body:	Workload

The actual compilation order is

Package spec:	Workload
Package body:	Workload
Package spec:	Periodic_Tasks
Package body:	Periodic_Tasks
Package spec:	Experiment
Package body:	Experiment
Main procedure:	Hartstone

```
--|-----------------------------------------------------------
--|
--|                Hartstone Benchmark, Version 1.0
--|
--| Unit Name: Hartstone
--|
--| Unit Type: Main Procedure Body
--|
--| Description:
--|  Controls a single Hartstone experiment. A Hartstone experiment consists
--|  of a series of individual tests, with the tests being performed by a
--|  set of tasks. The tasks are required to perform varying computational
--|  loads within hard-real-time deadlines. (The name Hartstone comes from
--|  HArd Real-Time and the fact that the computational load is provided by
--|  a variant of the Whetstone benchmark.) This main program activates the
--|  set of tasks and collects results from it.
--|
--|  As each test completes, its results are stored and may optionally
--|  be output at that time. Also, a check is made to see if the entire
--|  experiment has completed. If not, the next test in the series is
--|  started. On completion of the experiment, a summary of the results
--|  is output.
--|
--| Authors:
--|  Nelson Weiderman, Neal Altman, Patrick Donohoe, Ruth Shapiro,
--|  Software Engineering Institute,
--|  Carnegie Mellon University,
--|  Pittsburgh, PA 15213.
--|
--| References:
--|  Weiderman, N.,
--|    Hartstone: Synthetic Benchmark Requirements
--|      for Hard Real-Time Applications
--|    Technical Report CMU/SEI-89-TR-23,
--|    Software Engineering Institute, June 1989.
--|
--|  Donohoe, P., Shapiro, R., Weiderman, N.,
--|    Hartstone Benchmark User's Guide, Version 1.0
--|    Technical Report CMU/SEI-90-UG-1,
--|    Software Engineering Institute, March 1990.
--|
--|
--| Distribution and Copyright Notice:
--|
--|   Copyright (C) 1989 by the Carnegie Mellon University, Pittsburgh, PA.
--| The Software Engineering Institute (SEI) is a federally funded research
--| and development center established and operated by Carnegie Mellon
--| University (CMU). Sponsored by the U.S. Department of Defense under
--| contract F19628-85-C-0003, the SEI is supported by the services and
--| defense agencies, with the U.S. Air Force as the executive contracting
--| agent.
--|
--|   Permission to use, copy, modify, or distribute this software and its
--| documentation for any purpose and without fee is hereby granted,
--| provided that the above copyright notice appears in all copies and that
--| both that copyright notice and this permission notice appear in
--| supporting documentation. Further, the names Software Engineering
--| Institute or Carnegie Mellon University may not be used in advertising
--| or publicity pertaining to distribution of the software without
--| specific, written prior permission. CMU makes no claims or
--| representations about the suitability of this software for any purpose.
--| This software is provided "as is" and no warranty, express or implied,
--| is made by the SEI or CMU, as to the accuracy and functioning of the
--| program and related program material, nor shall the fact of distribution
--| constitute any such warranty. No responsibility is assumed by the SEI
--| or CMU in connection herewith.
--|
--|--------------------------------------------------------------

with Experiment;
with Periodic_Tasks;

with Calendar;
use  Calendar; -- To gain visibility for arithmetic operations on time
with System;

procedure Hartstone is

  pragma Priority(System.Priority'Last); -- Higher than any Hartstone task

  -- Variables to hold test parameters

  Test_Start_Time         : Calendar.Time;
  Test_Duration           : Duration;
  No_Of_Test_Repetitions  : Natural;
  Full_Output             : Boolean;

  Task_Frequencies        : Experiment.Frequency_Array;
  Task_Workloads          : Experiment.Workload_Array;
  No_Of_Extra_Tasks       : Experiment.Extra_Tasks_Range;
  Extra_Tasks             : array(1..Experiment.Max_Extra_Tasks)
                              of Periodic_Tasks.New_Task_Pointer;

  -- The following constant is added to the start time of a test to
  -- allow for task elaboration etc., particularly when new tasks
  -- are being added to the baseline set

  Settling_Time : constant := 5.0;

  -- Variables to hold test results

  Met_Deadlines      : Experiment.Deadlines_Array;
  Missed_Deadlines   : Experiment.Deadlines_Array;
  Skipped_Deadlines  : Experiment.Deadlines_Array;
  Cumulative_Late    : Experiment.Cumulative_Late_Array;


  procedure Start_Test is -- Activate all tasks at a common starting time

  begin
    Periodic_Tasks.T1.Start(Test_Start_Time,
                            Test_Duration,
                            Task_Frequencies(1),
                            Task_Workloads(1));

    Periodic_Tasks.T2.Start(Test_Start_Time,
                            Test_Duration,
                            Task_Frequencies(2),
                            Task_Workloads(2));

    Periodic_Tasks.T3.Start(Test_Start_Time,
                            Test_Duration,
                            Task_Frequencies(3),
                            Task_Workloads(3));

    Periodic_Tasks.T4.Start(Test_Start_Time,
                            Test_Duration,
                            Task_Frequencies(4),
                            Task_Workloads(4));
    Periodic_Tasks.T5.Start(Test_Start_Time,
                            Test_Duration,
                            Task_Frequencies(5),
                            Task_Workloads(5));

    for I in 1..No_Of_Extra_Tasks loop
      Extra_Tasks(I).Start(Test_Start_Time,
                           Test_Duration,
                           Task_Frequencies(Experiment.No_Of_Basic_Tasks + I),
                           Task_Workloads(Experiment.No_Of_Basic_Tasks + I));
    end loop;
  end Start_Test;

  procedure Stop_Test is -- Retrieve task results on completion of a test
  begin
    Periodic_Tasks.T1.Stop(Met_Deadlines(1),
                           Missed_Deadlines(1),
                           Skipped_Deadlines(1),
                           Cumulative_Late(1));

    Periodic_Tasks.T2.Stop(Met_Deadlines(2),
                           Missed_Deadlines(2),
                           Skipped_Deadlines(2),
                           Cumulative_Late(2));

    Periodic_Tasks.T3.Stop(Met_Deadlines(3),
                           Missed_Deadlines(3),
                           Skipped_Deadlines(3),
                           Cumulative_Late(3));

    Periodic_Tasks.T4.Stop(Met_Deadlines(4),
                           Missed_Deadlines(4),
                           Skipped_Deadlines(4),
                           Cumulative_Late(4));

    Periodic_Tasks.T5.Stop(Met_Deadlines(5),
                           Missed_Deadlines(5),
                           Skipped_Deadlines(5),
                           Cumulative_Late(5));

    for I in 1..No_Of_Extra_Tasks loop
      Extra_Tasks(I).Stop(Met_Deadlines(Experiment.No_Of_Basic_Tasks + I),
                          Missed_Deadlines(Experiment.No_Of_Basic_Tasks + I),
                          Skipped_Deadlines(Experiment.No_Of_Basic_Tasks + I),
                          Cumulative_Late(Experiment.No_Of_Basic_Tasks + I));
    end loop;
  end Stop_Test;

begin -- Hartstone

  -- Get some basic experiment parameters common to all tests

  Experiment.Initialize(Test_Duration,
                        No_Of_Test_Repetitions,
                        Full_Output);

  -- Perform the tests of the experiment until a pre-determined
  -- experiment-completion criterion is satisfied

  loop

    -- Retrieve the current test parameters
```

```
      Experiment.Get_Test(Task_Frequencies,
                          Task_Workloads,
                          No_Of_Extra_Tasks);

      -- If the current experiment requires it, create a new task

      if No_Of_Extra_Tasks > 0 then
        Extra_Tasks(No_Of_Extra_Tasks) := new Periodic_Tasks.New_Task;
      end if;

      -- Repeat each test a pre-determined number of times

      for I in 1..No_Of_Test_Repetitions loop

        Met_Deadlines      := (others => 0);
        Missed_Deadlines   := (others => 0);
        Skipped_Deadlines  := (others => 0);
        Cumulative_Late    := (others => 0.0);

        Test_Start_Time    := Calendar.Clock + Settling_Time;

        Start_Test;

        -- Delay the main program beyond the end of the test (add twice
        -- the longest period) so that the rendezvous calls to collect
        -- test results won't interfere with the tasks as they finish up

        delay Settling_Time +
              Test_Duration +
              2 * Duration(1.0 / Task_Frequencies(Task_Frequencies'First));

        Stop_Test;

        Experiment.Store_Test_Results(Met_Deadlines,
                                      Missed_Deadlines,
                                      Skipped_Deadlines,
                                      Cumulative_Late);

        if Full_Output then
          Experiment.Output_Test_Results;  -- Results of current test
        end if;
      end loop;

      exit when Experiment.Is_Complete;

    end loop;

    Experiment.Output_Summary_Results;  -- Summary of entire experiment

  end Hartstone;
```

```
--|----------------------------------------------------------
--|
--| Unit Name: Experiment
--|
--| Unit Type: Package Specification
--|
--| Description:
--|  Provides the interfaces for retrieving the characteristics of
--|  experiments and their constituent tests, storing and displaying
--|  test and experiment results, and checking for experiment completion.
--|
--|----------------------------------------------------------

package Experiment is

   -- Exported constants, types, and subtypes

   Benchmark : constant String := "Hartstone Benchmark, Version 1.0";

   No_Of_Basic_Tasks : constant Natural := 5;
   Max_Extra_Tasks   : constant Natural := 100;
   subtype Extra_Tasks_Range is Integer range -1..Max_Extra_Tasks;

   subtype Task_Number_Range is Natural
     range 1..No_Of_Basic_Tasks + Max_Extra_Tasks;

   type Frequency_Array        is array(Task_Number_Range) of Float;
   type Workload_Array         is array(Task_Number_Range) of Natural;
   type Work_Rate_Array        is array(Task_Number_Range) of Float;
   type Deadlines_Array        is array(Task_Number_Range) of Natural;
   type Cumulative_Late_Array  is array(Task_Number_Range) of Duration;

   --|
   --| Unit Name: Initialize
   --|
   --| Unit Type: Procedure Specification
   --|
   --| Description:
   --|  Retrieves test parameters which are common to all tests in
   --|  the experiment.
   --|
   --| Parameters:
   --|  Length_Of_Test: The duration of each test in the experiment,
   --|    measured in seconds.
   --|
   --|  No_Of_Repetitions: The number of times the current test is run
   --|    before the next test in the series is started.
   --|
   --|  Full_Output_Requested: If false, only summary results are
   --|    output at the end of the experiment. If true, results are
   --|    also output as each test repetition completes.
   --|

   procedure Initialize(Length_Of_Test        : out Duration;
                        No_Of_Repetitions     : out Positive;
                        Full_Output_Requested : out Boolean);


   --|
   --| Unit Name: Get_Test
   --|
   --| Unit Type: Procedure Specification
   --|
   --| Description:
```

```
--|  Retrieves the characteristics of the current test in a test series.
--|
--| Parameters:
--|  Frequencies: Each element of this array contains the frequency,
--|    in Hertz, of the corresponding Hartstone task.
--|
--|  Workloads: Each element of this array contains the workload,
--|    expressed in thousands of Whetstone instructions, of the
--|    corresponding Hartstone task.
--|
--|  Extra_Tasks: The total number of extra tasks to be exectuted along
--|    with the baseline set of Hartstone tasks, in a single test.
--|

procedure Get_Test (Frequencies : out Frequency_Array;
                    Workloads   : out Workload_Array;
                    Extra_Tasks : out Extra_Tasks_Range);


--|
--| Unit Name: Is_Complete
--|
--| Unit Type: Function Specification
--|
--| Description:
--|  Checks the completion criterion established for the experiment in
--|  progress.  Returns a Boolean value indicating whether (true) or not
--|  (false) the experiment is finished.
--|
--| Parameters: None
--|

function Is_Complete return Boolean;


--|
--| Unit Name: Store_Test_Results
--|
--| Unit Type: Procedure Specification
--|
--| Description:
--|  Stores the results of the current test so that they may be used
--|  to check for experiment completion and/or delivered as output.
--|
--| Parameters:
--|  Met: Each element of this array contains the number of times during
--|    the test that the corresponding Hartstone task successfully
--|    completed its workload before its next scheduled activation time.
--|
--|  Missed: Each element of this array contains the number of times during
--|    the test that the corresponding Hartstone task failed to complete
--|    its workload before its next scheduled activation time.
--|
--|  Skipped: Each element of this array contains the number of times during
--|    the test that the corresponding Hartstone task did not attempt to
--|    perform its workload for the scheduled activation time.
--|
--|  Amount_Late: Each element of this array contains the sum of the
--|    amounts by which the corresponding Hartstone task was late when
--|    it missed its deadlines.
--|

procedure Store_Test_Results (Met         : in Deadlines_Array;
                              Missed      : in Deadlines_Array;
                              Skipped     : in Deadlines_Array;
```

```
                                      Amount_Late : in Cumulative_Late_Array);

    --|
    --| Unit Name: Output_Test_Results
    --|
    --| Unit Type: Procedure Specification
    --|
    --| Description:
    --|   Outputs the results of the single test just completed.
    --|
    --| Parameters: None
    --|

    procedure Output_Test_Results;


    --|
    --| Unit Name: Output_Summary_Results
    --|
    --| Unit Type: Procedure Specification
    --|
    --| Description:
    --|   Outputs a summary of the results of an entire experiment.
    --|
    --| Parameters: None
    --|

    procedure Output_Summary_Results;

  end Experiment;
```

```
--|------------------------------------------------------------
--|
--| Unit Name: Experiment
--|
--| Unit Type: Package Body
--|
--| Description:
--|  The characteristics of four experiments for the Hartstone Periodic
--|  Harmonic (PH) test series are defined here. Also provided are the
--|  procedures and functions to retrieve individual test characteristics,
--|  store and display test results, check for completion of an experiment,
--|  and output a summary of the entire experiment.
--|
--|  An experiment consists of a series of tests. The tests are performed
--|  by a set of tasks. The transition from one test to the next in the
--|  series is achieved by increasing the computational load required of the
--|  task set. The four experiments defined here are:
--|
--|   Experiment 1: Increase the frequency of the highest-frequency task
--|
--|   Experiment 2: Scale up the frequencies of all the tasks
--|
--|   Experiment 3: Increase the workloads of all tasks
--|
--|   Experiment 4: Add new tasks to the baseline task set
--|
--|  When the computational load required of the periodic tasks exceeds the
--|  processor's capability they will start to miss their deadlines. They
--|  will shed load by skipping workload assignments in an effort to reach
--|  a point where a workload may again be attempted. Deadlines ignored
--|  during load-shedding are known as skipped deadlines. The completion
--|  conditions for an experiment are largely defined in terms of missed
--|  and skipped deadlines. An experiment completes when a test satisfies
--|  one of the following user-selected completion criteria:
--|
--|  (a) Any task in the task set has missed at least one deadline
--|      in the current test
--|  (b) The cumulative number of missed and skipped deadlines in the
--|      task set, for the current test, reaches a pre-set limit
--|  (c) The cumulative number of missed and skipped deadlines in the
--|      task set, for the current test, reaches a pre-set percentage
--|      of the total number of met + missed + skipped deadlines
--|  (d) The workload required of the task set is more than it could
--|      possibly achieve, i.e. when the requested workload is greater
--|      than the workload achievable by the benchmark in the absence
--|      of tasking. This is a default completion criterion for all
--|      experiments.
--|  (e) The maximum number of extra tasks has been added to the task
--|      set and deadlines still have not been missed or skipped. This
--|      is a default completion criterion for experiment 4. If this
--|      happens, the user should increase the value of the parameter
--|      representing the maximum number of tasks to be added.
--|
--|  Since this benchmark is primarily for embedded targets, no assumptions
--|  are made about the availability of host-target file I/O, or the ability
--|  to provide parameters to the executing benchmark interactively. It is
--|  assumed that the only code executing on the target system is the
--|  Hartstone benchmark and the Ada run-time system. Thus the experiment
--|  to be performed, the conditions under which it stops, and the
--|  characteristics of tests within the experiment are all defined here
--|  and are changed by manual editing of this package body. The part of
--|  the package that needs to be modified by users is small and explicitly
--|  indicated by comments. For any given experiment, the changed package
--|  body must be re-compiled, and the Hartstone benchmark re-linked and re-
--|  loaded into the target.
--|
--|-----------------------------------------------------------------


with Workload;

with Calendar;
with Text_IO;

package body Experiment is

  type Experiment_Type is (Experiment_1, Experiment_2,
                           Experiment_3, Experiment_4);

  type Completion_Type is (One_Unmet_Deadline,
                           Many_Unmet_Deadlines,
                           Percent_Unmet_Deadlines);

  -------------<<< START OF USER-MODIFIABLE SECTION >>>-------------

  -- Modify the next two strings to describe your compiler and target

  Compiler : constant String :=
             "XXX Host_System -> Target_System, release n.n";
  Target   : constant String :=
             "Target_System (m.n MHz)";

  -- Modify only the next two values to implement a particular
  -- experiment using the default parameters

  Which_Experiment      : constant Experiment_Type := Experiment_1;
  Completion_Criterion  : constant Completion_Type := Many_Unmet_Deadlines;


  -- Modify the parameters below ONLY if you wish to change the default
  -- characteristics of an experiment.

  -- Experiment characteristics:

  No_Of_Test_Repetitions : constant := 1;
  Full_Output    : constant Boolean  := True;  -- False => only output a summary
  Test_Duration  : constant Duration := 10.0;  -- Seconds

  -- Experiment completion criteria parameters:

  Unmet_Deadlines_Limit          : constant := 50;
  Percent_Unmet_Deadlines_Limit  : constant := 50.0;

  -- Task set characteristics:

  -- Bear in mind that the harmonic nature of the PH test series must be
  -- preserved and that rate-monotonic priorities for tasks depend on the
  -- task frequencies (higher-frequency task => higher priority, and vice
  -- versa). Also note that the frequency specified in the "others"
  -- choice must be the same as the third array element.
  Initial_Task_Frequencies : constant Frequency_Array :=
                              (2.0, 4.0, 8.0, 16.0, 32.0,
                               others => 8.0);

  -- The set of initial workloads provides each task with the same
  -- workload per second (frequency x workload). The workload specified
  -- in the "others" choice must be the same as the third array element.
  Initial_Task_Workloads    : constant Workload_Array :=
                                  (32, 16,  8,  4,  2,
                                   others => 8);

-- The frequency increment for the highest-frequency task in the basic
-- task set must be set equal to the frequency of the next-to-last task
-- in order to preserve the harmonic nature of the PH series task set
Frequency_Increment     : constant Float :=
                            Initial_Task_Frequencies(No_Of_Basic_Tasks - 1);
Workload_Increment      : constant Natural := 1;
Frequency_Scale_Factor  : constant Float := 0.1;

-------------<<< END OF USER-MODIFIABLE SECTION >>>-------------

type Test_State is
  record
    Test_Number           : Natural := 0;
    No_Of_Extra_Tasks     : Extra_Tasks_Range := -1;
    Total_No_Of_Tasks     : Natural := No_Of_Basic_Tasks;
    Task_Frequencies      : Frequency_Array := Initial_Task_Frequencies;
    Task_Workloads        : Workload_Array  := Initial_Task_Workloads;
    Met_Deadlines         : Deadlines_Array := (others => 0);
    Missed_Deadlines      : Deadlines_Array := (others => 0);
    Skipped_Deadlines     : Deadlines_Array := (others => 0);
    Cumulative_Late       : Cumulative_Late_Array := (others => 0.0);
    Task_Work_Rates       : Work_Rate_Array := (others => 0.0);
    Total_Rate_Requested  : Float := 0.0;
    Total_Rate_Achieved   : Float := 0.0;
  end record;

Initial_Test          : Test_State;
Current_Test          : Test_State;
First_Failed_Test     : Test_State;
Last_Successful_Test  : Test_State;

Total_Met_Deadlines   : Natural := 0;
Total_Unmet_Deadlines : Natural := 0;
Raw_Speed             : Float   := 0.0;
Experiment_Step_Size  : Float   := 0.0;

-----------------------------oOo-----------------------------

--|
--| Unit Name: Compute_Raw_Speed
--|
--| Unit Type: Procedure Body
--|
--| Description:
--|  This local procedure provides a "calibration" of the computational
--|  load required of Hartstone tasks. A synthetic workload for each task
--|  is provided by a variant of the Whetstone benchmark. This procedure
--|  computes the raw speed of the Small_Whetstone benchmark, in the
--|  absence of tasking, by determining how many thousands of Whetstone
--|  instructions (Kilo-Whetstones) per second it can execute. Raw
--|  speed is expressed in Kilo-Whetstone Instructions Per Second (KWIPS).
--|  The performance of the Hartstone task set will be measured against
--|  this non-tasking computation.
--|
--|  The accuracy of this timing measurement will depend on the resolution
--|  of Calendar.Clock.
--|

procedure Compute_Raw_Speed is

  use Calendar; -- To achieve visibility of operations on Time values
```

```
    Iterations   : constant := 10_000; -- Thousands of Whetstone instructions
    Start_Time   : Calendar.Time;
    Finish_Time  : Calendar.Time;

  begin

    -- The number of loop iterations depends on the desired timing accuracy
    -- and the accuracy of Calendar.Clock. For example, to achieve an
    -- accuracy of one microsecond with a ten-millisecond Clock, the loop
    -- should iterate 10000 times. Note that to achieve a constant overhead
    -- the Small_Whetstone procedure is called repeatedly with a value of 1,
    -- representing one Kilo_Whetstone. This is also how Hartstone tasks do
    -- their workloads; the overhead of the procedure call is part of a
    -- task's overall execution time.

    Start_Time := Calendar.Clock;
    for I in 1..Iterations loop
      Workload.Small_Whetstone(1);
    end loop;
    Finish_Time := Calendar.Clock;
    Raw_Speed := Float(Iterations) / Float(Finish_Time - Start_Time);

  end Compute_Raw_Speed;

--------------------------------oOo--------------------------------

  --|
  --| Unit Name: Initialize
  --|
  --| Unit Type: Procedure Body
  --|
  --| Description:
  --|   Retrieves the basic test parameters common to all tests in an
  --|   experiment, i.e. the duration of a test, the number of times
  --|   the same test is to be repeated, and whether or not the results
  --|   of a test should be output when the test completes. (A summary
  --|   of the entire experiment will always be output.) Also computes
  --|   the raw (non-tasking) speed of the benchmark and the step size
  --|   of the experiment.
  --|

  procedure Initialize(Length_Of_Test        : out Duration;
                       No_Of_Repetitions     : out Positive;
                       Full_Output_Requested : out Boolean) is
  begin

    -- "Calibrate" the Hartstone benchmark by measuring the speed
    -- of the synthetic workload in the absence of tasking

    Compute_Raw_Speed;

    -- Determine the step size of the experiment. "Step size" is a measure
    -- of the extra work requested of the task set when the next test in
    -- a series is derived from the current test. It is expressed as a
    -- percentage of the raw speed. It varies from experiment to experiment
    -- but remains constant for a specific experiment.

    case Which_Experiment is

      when Experiment_1 =>

        -- The step size of Experiment 1 is equal to the amount of extra
        -- work given to the highest-frequency task divided by the raw speed
        Experiment_Step_Size := 100.0 * (Frequency_Increment *
          Float(Initial_Task_Workloads(No_Of_Basic_Tasks))) / Raw_Speed;

      when Experiment_2 =>

        -- The step size of Experiment 2 is equal to the amount of extra
        -- work given to all the tasks divided by the raw speed

        for I in 1..No_Of_Basic_Tasks loop
          Experiment_Step_Size := Experiment_Step_Size +
            100.0 * (Frequency_Scale_Factor * Initial_Task_Frequencies(I) *
              Float(Initial_Task_Workloads(I))) / Raw_Speed;
        end loop;

      when Experiment_3 =>

        -- The step size of Experiment 3 is equal to the amount of extra
        -- work given to all the tasks divided by the raw speed

        for I in 1..No_Of_Basic_Tasks loop
          Experiment_Step_Size := Experiment_Step_Size +
            100.0 * (Float(Workload_Increment) * Initial_Task_Frequencies(I)) /
              Raw_Speed;
        end loop;

      when Experiment_4 =>

        -- The step size of Experiment 4 is equal to the amount of work
        -- performed by a new task divided by the raw speed.

        Experiment_Step_Size := 100.0 *
          (Initial_Task_Frequencies(No_Of_Basic_Tasks + 1) *
            Float(Initial_Task_Workloads(No_Of_Basic_Tasks + 1))) / Raw_Speed;

    end case;

    Length_Of_Test          := Test_Duration;
    No_Of_Repetitions       := No_Of_Test_Repetitions;
    Full_Output_Requested   := Full_Output;

  end Initialize;
```

------------------------------oOo------------------------------

```
  --|
  --| Unit Name: Get_Test
  --|
  --| Unit Type: Procedure Body
  --|
  --| Description:
  --|  Retrieves the frequencies and workloads to be assigned to the Hartstone
  --|  task set for the current test. Also retrieves a count of the number
  --|  of extra tasks to be executed along with the baseline set (if required
  --|  by the experiment) in the current test. It is this procedure which
  --|  manages the transition from one test to the next in an experiment.
  --|

  procedure Get_Test(Frequencies  : out Frequency_Array;
                     Workloads    : out Workload_Array;
                     Extra_Tasks  : out Extra_Tasks_Range) is

  begin

    -- Update parameters for this test, in accordance with current experiment
    case Which_Experiment is

      when Experiment_1 =>

        -- Increment frequency of highest-frequency basic task

        Current_Test.Task_Frequencies(No_Of_Basic_Tasks) :=
          Initial_Task_Frequencies(No_Of_Basic_Tasks) +
            Float(Current_Test.Test_Number) * Frequency_Increment;

      when Experiment_2 =>

        -- Scale up frequencies of all basic tasks

        for I in 1..No_Of_Basic_Tasks loop
          Current_Test.Task_Frequencies(I) :=
            Initial_Task_Frequencies(I) + Float(Current_Test.Test_Number) *
              Frequency_Scale_Factor * Initial_Task_Frequencies(I);

        end loop;

      when Experiment_3 =>

        -- Increment workloads of all basic tasks

        for I in 1..No_Of_Basic_Tasks loop
          Current_Test.Task_Workloads(I) := Initial_Task_Workloads(I) +
            Current_Test.Test_Number * Workload_Increment;
        end loop;

      when Experiment_4 =>

        -- For each test, add a new task (dynamically
        -- created in the main program) to the task set

        Current_Test.No_Of_Extra_Tasks := Current_Test.No_Of_Extra_Tasks + 1;
        Current_Test.Total_No_Of_Tasks := No_Of_Basic_Tasks +
          Current_Test.No_Of_Extra_Tasks;

    end case;

    Current_Test.Test_Number := Current_Test.Test_Number + 1;

    -- Return task characteristics for current test

    Frequencies := Current_Test.Task_Frequencies;
    Workloads   := Current_Test.Task_Workloads;
    Extra_Tasks := Current_Test.No_Of_Extra_Tasks;

  end Get_Test;

-----------------------------oOo-----------------------------

  --|
  --| Unit Name: Is_Complete
  --|
  --| Unit Type: Function Body
  --|
  --| Description:
  --|  Checks the completion criterion established for the experiment in
  --|  progress. Returns a Boolean value indicating whether (true) or not
  --|  (false) the experiment is finished. The completion criteria are
  --|  defined in terms of the maximum allowed number of unmet deadlines
  --|  for the Hartstone task set.
  --|
function Is_Complete return Boolean is

begin

   -- Check the default completion criteria.  These are: stop any experiment
   -- when the work rate requested of the task set exceeds that achievable
   -- by the non-tasking benchmark (raw speed), and stop Experiment 4 when
   -- the maximum number of extra tasks have been added, whether or not
   -- deadlines have been missed.

   if Current_Test.Total_Rate_Requested >= Raw_Speed or
     Current_Test.No_Of_Extra_Tasks = Max_Extra_Tasks then
     return True;
   end if;

   -- Check the user-specified completion criterion

   case Completion_Criterion is

     when One_Unmet_Deadline =>
       return Total_Unmet_Deadlines >= 1;

     when Many_Unmet_Deadlines =>
       return Total_Unmet_Deadlines >= Unmet_Deadlines_Limit;

     when Percent_Unmet_Deadlines =>

       return (Float(Total_Unmet_Deadlines) /
         Float(Total_Met_Deadlines + Total_Unmet_Deadlines)) * 100.0 >=
           Percent_Unmet_Deadlines_Limit;

   end case;

end Is_Complete;
```

------------------------------oOo------------------------------

```
--|
--| Unit Name: Store_Test_Results
--|
--| Unit Type: Procedure Body
--|
--| Description:
--|   Stores the results of the current test so that they may be used
--|   to check for experiment completion and/or delivered as output.
--|   Also saves the results of the highest test in the series with no
--|   missed or skipped deadlines, the results at the time deadlines
--|   were first missed or skipped, and the results of the first test
--|   in the series.
--|

procedure Store_Test_Results (Met         : in Deadlines_Array;
                              Missed      : in Deadlines_Array;
                              Skipped     : in Deadlines_Array;
                              Amount_Late : in Cumulative_Late_Array) is

begin

   -- Store the results provided in the call

   Current_Test.Met_Deadlines      := Met;
   Current_Test.Missed_Deadlines   := Missed;
   Current_Test.Skipped_Deadlines  := Skipped;
   Current_Test.Cumulative_Late    := Amount_Late;
   -- Derived results

   Current_Test.Total_Rate_Requested := 0.0; -- Task set's requested work rate
   Current_Test.Total_Rate_Achieved  := 0.0; -- Task set's achieved work rate
   Total_Met_Deadlines   := 0;
   Total_Unmet_Deadlines := 0;

   for I in 1..Current_Test.Total_No_Of_Tasks loop

     -- Calculate the rates at which tasks are required to do their workloads

     Current_Test.Task_Work_Rates(I) := (Current_Test.Task_Frequencies(I) *
       Float(Current_Test.Task_Workloads(I)));

     -- The task set's requested work rate is the sum of the tasks' work rates

     Current_Test.Total_Rate_Requested := Current_Test.Total_Rate_Requested +
       Current_Test.Task_Work_Rates(I);

     -- Calculate the rate at which the task set's workload was actually done

     Current_Test.Total_Rate_Achieved := Current_Test.Total_Rate_Achieved +
       (Float(Current_Test.Met_Deadlines(I) *
         Current_Test.Task_Workloads(I)) / Float(Test_Duration));

     Total_Met_Deadlines := Total_Met_Deadlines +
                            Current_Test.Met_Deadlines(I);

     Total_Unmet_Deadlines := Total_Unmet_Deadlines +
                              Current_Test.Skipped_Deadlines(I) +
                              Current_Test.Missed_Deadlines(I);

   end loop;

   -- If the current test hasn't missed/skipped any deadlines yet
   -- then record its state as the best result so far, otherwise,
   -- if the current test is the first to miss/skip deadlines,
   -- record the state of the task set at the time of the miss/skip.

   if Total_Unmet_Deadlines = 0 then
     Last_Successful_Test := Current_Test;
   elsif First_Failed_Test.Test_Number = 0 then
     First_Failed_Test := Current_Test;
   end if;

   -- Save the initial (baseline) test results for the summary

   if Current_Test.Test_Number = 1 then
     Initial_Test := Current_Test;
   end if;

 end Store_Test_Results;
```

------------------------------oOo------------------------------

```
 --|
 --| Unit Name: Put_Results
 --|
 --| Unit Type: Procedure Body
 --|
 --| Description:
 --|   For the given test, outputs the name of the experiment, the
 --|   completion criterion, the characteristics of the task set (its
 --|   frequencies, workloads, and utilization) and the results
```

```
--| achieved by the task set (met, missed, and skipped deadlines).
--| Also ouputs the raw (non-tasking) speed and the experiment
--| step size. Utilization is defined as the workload execution
--| rate expressed as a percentage of the raw speed. The step
--| size is the increase in utilization required of the task set
--| when performing the successor to the current test. It is
--| the "resolution" of the workload utilization.
--|

procedure Put_Results(Test : in Test_State) is

  package Flt_IO      is new Text_IO.Float_IO(Float);
  package Int_IO      is new Text_IO.Integer_IO(Integer);
  package Duration_IO is new Text_IO.Fixed_IO(Duration);
  use Text_IO;

begin

  New_Line;
  Put_Line("========================================================================")

  New_Line;
  Put_Line("Experiment:    " & Experiment_Type'Image(Which_Experiment));

  Put("Completion on: ");
  case Completion_Criterion is
    when One_Unmet_Deadline =>
      Put_Line("Miss/skip at least one deadline");
    when Many_Unmet_Deadlines =>
      Put_Line("Miss/skip" & Integer'Image(Unmet_Deadlines_Limit) &
               " deadlines");
    when Percent_Unmet_Deadlines =>
      Put("Miss/skip ");
      Flt_IO.Put(Float(Percent_Unmet_Deadlines_Limit), 3, 1, 0);
      Put_Line(" percent of deadlines");
  end case;

  New_Line;
  Put("Raw speed in Kilo-Whetstone Instructions Per Second (KWIPS): ");
  Flt_IO.Put(Raw_Speed, 4, 2, 0);
  New_Line;

  New_Line;
  Put_Line("Test" & Integer'Image(Test.Test_Number) & " characteristics:");
  New_Line;

  Put_Line("   Task   Frequency   Kilo-Whets   Kilo-Whets   Requested Workload");
  Put_Line("    No.    (Hertz)    per period   per second      Utilization");

  for I in 1..Test.Total_No_Of_Tasks loop

    -- Task number
    Set_Col(5);
    Int_IO.Put(I, 2);

    -- Task frequency
    Set_Col(11);
    Flt_IO.Put(Test.Task_Frequencies(I), 4, 2, 0);

    -- Kilo Whetstones per period
    Set_Col(25);
    Int_IO.Put(Test.Task_Workloads(I), 4);

    -- Kilo Whetstone Instructions Per Second
    Set_Col(37);
```

```
   Flt_IO.Put(Test.Task_Frequencies(I) * Float(Test.Task_Workloads(I)), 4, 2, 0);

   -- Requested KWIPS as a % of the raw speed
   Set_Col(53);
   Flt_IO.Put((100.0 * Test.Task_Work_Rates(I) / Raw_Speed), 3, 2, 0);
   Put_Line(" %");

end loop;

Set_Col(37);
Put("-------");
Set_Col(53);
Put_Line("--------");
Set_Col(37);
Flt_IO.Put(Test.Total_Rate_Requested, 4, 2, 0);
Set_Col(53);
Flt_IO.Put((100.0 * Test.Total_Rate_Requested / Raw_Speed), 3, 2, 0);
Put_Line(" %");

New_Line;
Put("Experiment step size: ");
Flt_IO.Put(Experiment_Step_Size, 3, 2, 0);
Put_Line(" %");

New_line;
Put_Line("-------------------------------------------------------------------------")
New_Line;
Put_Line("Test" & Integer'Image(Test.Test_Number) & " results:");

New_Line;
Put("Test duration (seconds): ");
Duration_IO.Put(Test_Duration, 3, 1, 0);
New_Line;

New_Line;
Put_Line("   Task    Period      Met       Missed      Skipped      Average");
Put_Line("   No.   in msecs  Deadlines   Deadlines   Deadlines   Late (msec)");

for I in 1..Test.Total_No_Of_Tasks loop

   Set_Col(5);
   Int_IO.Put(I, 2); -- Task number

   Set_Col(11);
   Flt_IO.Put((1000.0 / Test.Task_Frequencies(I)), 4, 3, 0); -- Task period

   Set_Col(23);
   Int_IO.Put(Test.Met_Deadlines(I), 5);

   Set_Col(35);
   Int_IO.Put(Test.Missed_Deadlines(I), 5);

   Set_Col(47);
   Int_IO.Put(Test.Skipped_Deadlines(I), 5);

   Set_Col(58);
   if (Test.Missed_Deadlines(I) > 0) then
     Flt_IO.Put(1000.0 * Float(Test.Cumulative_Late(I)) /
       Float(Test.Missed_Deadlines(I)), 5, 3, 0); -- Average late amount
   else
     Flt_IO.Put(1000.0 * Float(Test.Cumulative_Late(I)), 5, 3, 0);
   end if;
   New_Line;

end loop;
```

```
    New_line;
    Put_Line ("===========================================================================")
    New_Line;

  end Put_Results;

----------------------------oOo----------------------------

  --|
  --| Unit Name: Output_Test_Results
  --|
  --| Unit Type: Procedure Body
  --|
  --| Description:
  --|   Output the results for the test just completed.
  --|

  procedure Output_Test_Results is

  begin

    Put_Results(Current_Test);

  end Output_Test_Results;

----------------------------oOo----------------------------

  --|
  --| Unit Name: Output_Summary_Results
  --|
  --| Unit Type: Procedure Body
  --|
  --| Description:
  --|   Outputs a summary of the results of an entire experiment. The summary
  --|   includes the results of the first test, the best test with no missed
  --|   or skipped deadlines, the test where deadlines were first missed, and
  --|   the final test run.
  --|

  procedure Output_Summary_Results is

    package Flt_IO is new Text_IO.Float_IO(Float);
    package Int_IO is new Text_IO.Integer_IO(Integer);
    use Text_IO;

    Test : Test_State; -- For output of overall summary of best test result

  begin

    if Full_Output then
      New_Page;
    end if;

    New_Line(2);
    Put_Line("                      HARTSTONE BENCHMARK SUMMARY RESULTS");
    New_Line;

    -- Output the results of the key tests. Because some run-time systems
    -- have problems outputting the volume of summary data, a delay has
    -- been inserted between each summary to slow down the output.

    delay 5.0;

    New_Line;
```

```
Put_Line("Baseline test:  ");
Put_Results(Initial_Test);
New_Page;

delay 5.0;

New_Line(2);
Put_Line("Last test with no missed/skipped deadlines: ");
if Last_Successful_Test.Test_Number > Initial_Test.Test_Number then
  Put_Results(Last_Successful_Test);
  New_Page;
elsif Last_Successful_Test.Test_Number = Initial_Test.Test_Number then
  Put_Line("   See preceding summary of test" &
    Integer'Image(Initial_Test.Test_Number));
  New_Line(2);
else
  Put_Line("   Not applicable");
  New_Line(2);
end if;

delay 5.0;

New_Line(2);
Put_Line("Test when deadlines first missed/skipped: ");
if First_Failed_Test.Test_Number > Initial_Test.Test_Number then
  Put_Results(First_Failed_Test);
  New_Page;
elsif First_Failed_Test.Test_Number = Initial_Test.Test_Number then
  Put_Line("   See preceding summary of test" &
    Integer'Image(Initial_Test.Test_Number));
  New_Line(2);
else
  Put_Line("   Not applicable");
  New_Line(2);
end if;

delay 5.0;

New_Line(2);
Put_Line("Final test performed: ");
if Current_Test.Test_Number = Initial_Test.Test_Number or
  Current_Test.Test_Number   = Last_Successful_Test.Test_Number or
  Current_Test.Test_Number   = First_Failed_Test.Test_Number then
  Put_Line("   See preceding summary of test" &
    Integer'Image(Current_Test.Test_Number));
  New_Line(2);
else
  Put_Results(Current_Test);
  New_Page;
end if;

-- Output "executive summary" of the best result for the compiler & target

delay 5.0;

New_Line(2);
Put_Line("==================  ==============================================");

New_Line;
Put_Line("Benchmark : " & Benchmark);
Put_Line("Compiler  : " & Compiler);
Put_Line("Target    : " & Target);

New_Line;
Put_Line("Characteristics of best test for this experiment: ");
Put_Line("(no missed/skipped deadlines)");
New_Line;

Test := Last_Successful_Test;
if Test.Test_Number = 0 then
  Put_Line("   Not applicable");
else
  Put("   Test" & Integer'Image(Test.Test_Number) & " of Experiment ");
  case Which_Experiment is
    when Experiment_1 =>
      Put_Line("1");
    when Experiment_2 =>
      Put_Line("2");
    when Experiment_3 =>
      Put_Line("3");
    when Experiment_4 =>
      Put_Line("4");
  end case;

  New_Line;
  Put("   Raw (non-tasking) benchmark speed in KWIPS: ");
  Flt_IO.Put(Raw_Speed, 4, 2, 0);
  New_Line;

  New_Line;
  Put_Line("   Full task set:");
  New_Line;
  Put_Line("      Total      Deadlines        Task Set       Total");
  Put_Line("      Tasks      Per Second      Utilization     KWIPS");

  -- Total tasks = no. of baseline tasks + any extra tasks
  Set_Col(8);
  Int_IO.Put(Test.Total_No_Of_Tasks, 3);

  -- Total deadlines per second is indicator of task switching overhead
  Set_Col(19);
  declare
    Sum : Float := 0.0; -- Total deadlines per sec is sum of frequencies
  begin
    for I in 1..Test.Total_No_Of_Tasks loop
      Sum := Sum + Test.Task_Frequencies(I);
    end loop;
    Flt_IO.Put(Sum, 4, 2, 0);
  end;

  -- Task set utilization
  Set_Col(34);
  Flt_IO.Put((100.0 * Test.Total_Rate_Requested / Raw_Speed), 3, 2, 0);
  Put(" %");

  -- Task set workload rate in Kilo-Whetstone Instructions Per Second
  Set_Col(48);
  Flt_IO.Put(Test.Total_Rate_Requested, 4, 2, 0);
  New_Line;

  New_Line;
  Put_Line("   Highest-frequency task:");
  New_Line;
  Put_Line("      Period     Deadlines         Task          Task ");
  Put_Line("      (msec)     Per Second      Utilization     KWIPS");

  -- Task period
  Set_Col(6);
  Flt_IO.Put(1000.0 / Test.Task_Frequencies(No_Of_Basic_Tasks), 3, 3, 0);
```

```
      -- Total deadlines per second
      Set_Col(19);
      Flt_IO.Put(Test.Task_Frequencies(No_Of_Basic_Tasks), 4, 2, 0);

      -- Task utilization
      Set_Col(34);
      Flt_IO.Put((100.0 *
        Test.Task_Work_Rates(No_Of_Basic_Tasks) / Raw_Speed), 3, 2, 0);
      Put(" %");

      -- Task speed in KWIPS
      Set_Col(48);
      Flt_IO.Put(Test.Task_Work_Rates(No_Of_Basic_Tasks), 4, 2, 0);
      New_Line;

      New_Line;
      Put("   Experiment step size: ");
      Flt_IO.Put(Experiment_Step_Size, 3, 2, 0);
      Put_Line(" %");

    end if;

    New_line;
    Put_Line("==========================================================================")

    New_Line(2);
    Put_Line("               END OF HARTSTONE BENCHMARK SUMMARY RESULTS");
    New_Line;

  end Output_Summary_Results;

end Experiment;
```

```
--|-----------------------------------------------------------
--|
--| Unit Name: Periodic_Tasks
--|
--| Unit Type: Package Specification
--|
--| Description:
--|   This is the basic set of independent periodic tasks that performs a
--|   test in a Hartstone test series. Each task has a specific frequency
--|   and workload. The goal of each task is to complete its workload within
--|   its period. The deadline for completion of the workload is the next
--|   scheduled activation time of the task. For any given period, a task
--|   will either meet or miss its deadline; at the end of the test each task
--|   will report the total number of deadlines it met and missed, together
--|   with the cumulative late amount by which deadlines were missed. To
--|   enable a task to continue past the point at which deadlines are first
--|   missed, tasks "reset" themselves by skipping one or more workload
--|   assignments until they reach a point where a workload may again be
--|   attempted. This process, called load-shedding, allows the failure
--|   pattern of tasks to be studied when the task results are collected.
--|
--|   Each task is assigned a priority according to the rate-monotonic
--|   scheduling algorithm: higher-frequency tasks are given higher
--|   priorities than lower-frequency tasks. Because Ada task priorities
--|   are statically assigned, each task in the baseline Hartstone task
--|   set with a unique priority is explicitly named.
--|
--|   One of the Hartstone experiments requires the addition of new tasks.
--|   These all have the same characteristics as a single specified task in
--|   the basic task set. In particular, they will all have the same priority,
--|   so a task type definition, provided below, can be used to define them.
--|
--| References:
--|   Liu, C. L., and Layland, J. W.,
--|     Scheduling Algorithms for Multiprogramming
--|       in a Hard-Real-Time Environment.
--|     Journal of the Association for Computing Machinery,
--|       Vol. 20, No.1, January 1973, pp. 46-41.
--|
--|   Sha, L., and Goodenough, J.,
--|     Real-Time Scheduling Theory and Ada.
--|     Technical Report CMU/SEI-89-TR-14,
--|     Software Engineering Institute, April 1989.
--|
--|-----------------------------------------------------------

with System;
with Calendar;

package Periodic_Tasks is

  Task_Workload_Failure: exception; -- Raised if Small_Whetstone fails

  -- T1 = lowest-priority task
  T1_Priority : constant System.Priority := System.Priority'First;
  T2_Priority : constant System.Priority := T1_Priority + 1;
  T3_Priority : constant System.Priority := T1_Priority + 2;
  T4_Priority : constant System.Priority := T1_Priority + 3;
  T5_Priority : constant System.Priority := T1_Priority + 4;
  -- T5 = highest-priority task

  New_Task_Priority : constant System.Priority := T3_Priority;
```

---------------------------oOo---------------------------

```
--|
--| Unit Name: T1 .. T5
--|
--| Unit Type: Task Specification
--|
--| Description:
--|   Periodic task to perform an assigned workload at a specific frequency.
--|
--| Parameters:
--|   Entry Start: Provides the input test parameters.
--|
--|     Test_Start_Time: The time at which the task starts performing
--|       its requested workload.
--|
--|     Test_Duration: The length of the test, in seconds.
--|
--|     Task_Frequency: The number of times per second the task is
--|       required to perform its requested workload.
--|
--|     Task_Workload: The amount of work required of the task, expressed
--|       as a number of Kilo-Whetstone instructions.  A variant of the
--|       Whetstone benchmark provides the computational load for the task.
--|
--|   Entry Stop: Allows the caller to retrieve test results from the task.
--|
--|     Task_Met_Deadlines: The number of times during the test that the
--|       task successfully completed its workload before its next scheduled
--|       activation time.
--|
--|     Task_Missed_Deadlines: The number of times during the test that the
--|       task failed to complete its workload before its next scheduled
--|       activation time.
--|
--|     Task_Skipped_Deadlines: The number of scheduled activation times
--|       which were not performed because of a previously missed deadline.
--|
--|     Task_Cumulative_Late: The sum of the amounts by which the task was
--|       late when it missed deadlines.
--|

task T1 is
  pragma Priority(T1_Priority);

  entry Start(Test_Start_Time : in Calendar.Time;
              Test_Duration   : in Duration;
              Task_Frequency  : in Float;
              Task_Workload   : in Natural);

  entry Stop(Task_Met_Deadlines     : out Natural;
             Task_Missed_Deadlines  : out Natural;
             Task_Skipped_Deadlines : out Natural;
             Task_Cumulative_Late   : out Duration);
end T1;
```

----------------------------oOo----------------------------

```
task T2 is
  pragma Priority(T2_Priority);

  entry Start(Test_Start_Time : in Calendar.Time;
              Test_Duration   : in Duration;
              Task_Frequency  : in Float;
              Task_Workload   : in Natural);
  entry Stop(Task_Met_Deadlines     : out Natural;
             Task_Missed_Deadlines  : out Natural;
             Task_Skipped_Deadlines : out Natural;
             Task_Cumulative_Late   : out Duration);
end T2;
```

------------------------------oOo------------------------------

```
task T3 is
  pragma Priority(T3_Priority);

  entry Start(Test_Start_Time : in Calendar.Time;
              Test_Duration   : in Duration;
              Task_Frequency  : in Float;
              Task_Workload   : in Natural);

  entry Stop(Task_Met_Deadlines     : out Natural;
             Task_Missed_Deadlines  : out Natural;
             Task_Skipped_Deadlines : out Natural;
             Task_Cumulative_Late   : out Duration);
end T3;
```

------------------------------oOo------------------------------

```
task T4 is
  pragma Priority(T4_Priority);

  entry Start(Test_Start_Time : in Calendar.Time;
              Test_Duration   : in Duration;
              Task_Frequency  : in Float;
              Task_Workload   : in Natural);

  entry Stop(Task_Met_Deadlines     : out Natural;
             Task_Missed_Deadlines  : out Natural;
             Task_Skipped_Deadlines : out Natural;
             Task_Cumulative_Late   : out Duration);
end T4;
```

------------------------------oOo------------------------------

```
task T5 is
  pragma Priority(T5_Priority);

  entry Start(Test_Start_Time : in Calendar.Time;
              Test_Duration   : in Duration;
              Task_Frequency  : in Float;
              Task_Workload   : in Natural);

  entry Stop(Task_Met_Deadlines     : out Natural;
             Task_Missed_Deadlines  : out Natural;
             Task_Skipped_Deadlines : out Natural;
             Task_Cumulative_Late   : out Duration);
end T5;
```

------------------------------oOo------------------------------

```
task type New_Task is
  pragma Priority(New_Task_Priority);

  entry Start(Test_Start_Time : in Calendar.Time;
              Test_Duration   : in Duration;
              Task_Frequency  : in Float;
              Task_Workload   : in Natural);

  entry Stop(Task_Met_Deadlines     : out Natural;
                    Task_Missed_Deadlines   : out Natural;
                    Task_Skipped_Deadlines  : out Natural;
                    Task_Cumulative_Late    : out Duration);
    end New_Task;

    type New_Task_Pointer is access New_Task;

  end Periodic_Tasks;
```

```
--|-----------------------------------------------------------------
--|
--| Unit Name:  Periodic_Tasks
--|
--| Unit Type:  Package Body
--|
--| Description:
--|   See the description in the package specification and the description
--|   of the local procedure below.
--|
--|   Because Ada priorities are assigned statically, the unique priorities
--|   of the baseline task set are assigned to explicitly-named tasks via the
--|   Priority pragma in the tasks' specifications.  It is not possible, for
--|   example, to define an array of baseline tasks whose elements are of a
--|   single task type, and whose priorities can be assigned at run-time.
--|   For similar reasons, a generic template is also not an option.  Thus the
--|   five tasks in the baseline set are task objects with distinct names and
--|   priorities, hence the "duplication" of code below.  The amount of text
--|   duplicated is reduced somewhat by putting actions common to all tasks in
--|   a procedure.
--|
--|-----------------------------------------------------------------

with Workload;

with Calendar;

package body Periodic_Tasks is

    --|
    --| Unit Name: Do_Work
    --|
    --| Unit Type: Procedure Body
    --|
    --| Description:
    --|   Local procedure to do the work of a Hartstone task.  Task periodicity
    --|   is implemented using the delay statement (expiry of a delay is assumed
    --|   to be pre-emptive) and Calendar.Clock.  The Small_Whetstone procedure,
    --|   a variant of the composite synthetic benchmark, provides the task
    --|   computational workload.  The workload is expressed in thousands of
    --|   Whetstone instructions (Kilo-Whetstones) and the rate of doing work
    --|   is measured in Kilo-Whetstone Instructions Per Second (KWIPS).
    --|   The deadline for completion of the workload during a task's period is
    --|   the next scheduled activation time of the task.  Late completion of
    --|   the workload is defined as a missed deadline.  Tasks continue to run
    --|   after deadlines are missed by skipping workload assignments until
    --|   they reach a point where a workload may again be attempted.  This
    --|   process is called load-shedding and any deadlines ignored during it
    --|   are recorded as skipped deadlines.
    --|
    --|   This procedure is based on the drift-free periodic procedure exhibited
    --|   in section 9.6 of the Ada Language Reference Manual.  To avoid problems
    --|   of cumulative error with the fixed-point type Duration, computations
    --|   are performed in floating-point arithmetic and only converted to
    --|   Duration in the actual delay statement.  Calculations involving the
    --|   type Time are also done in floating-point, by extracting the seconds
    --|   portion of the Time value (a private type) returned by Calendar.Clock
    --|   and converting it from Day_Duration to Float.  Because only the seconds
    --|   portion is used, the test duration should not cross a midnight
    --|   boundary; the Day_Duration value returned by Calendar.Seconds becomes
    --|   zero after 24 hours.
    --|
    --|   A number of implementation-dependent features are present in this
    --|   procedure: the accuracy of the Duration expression used in the delay
--| statement (depends on Duration'Small), the resolution of the delay
--| statement itself (the actual delay may be much larger than the
--| requested delay), and the resolution of Calendar.Clock (a coarse
--| resolution means that a coarse value will be used as the expression
--| in the delay statement, thereby resulting in a flawed implementation
--| of task periodicity).
--|

procedure Do_Work (Test_Start_Time        : in     Calendar.Time;
                   Test_Duration          : in     Duration;
                   Task_Frequency         : in     Float;
                   Task_Workload          : in     Natural;
                   Task_Met_Deadlines     :    out Natural;
                   Task_Missed_Deadlines  :    out Natural;
                   Task_Skipped_Deadlines :    out Natural;
                   Task_Cumulative_Late   :    out Duration) is

  use Calendar;  -- Make operators for Time and Duration calculations visible

  Finish_Time : Float := Float(Calendar.Seconds(Test_Start_Time +
                                                Test_Duration));
  Period      : Float := Float(1.0 / Task_Frequency);
  Next_Start  : Float := Float(Calendar.Seconds(Test_Start_Time));
  Next_Delay  : Float := 0.0;

  Met_Deadlines      : Natural := 0;
  Missed_Deadlines   : Natural := 0;
  Skipped_Deadlines  : Natural := 0;
  Cumulative_Late    : Float   := 0.0;  -- Sum of missed deadline late amounts

  Now                : Float   := 0.0;  -- Will be used during load shedding
  Old_Met_Deadlines  : Natural := 0;    -- Will be used during load shedding

begin -- Do_Work

  while Next_Start < Finish_Time loop

    Next_Delay := Next_Start - Float( Calendar.Seconds(Calendar.Clock) );

    if Next_Delay >= 0.0 then

      -- A positive delay computation means either that the task completed
      -- its last workload on time or that the load-shedding to compensate
      -- for the last missed deadline was successful

      delay Duration(Next_Delay);
      for I in 1..Task_Workload loop
        Workload.Small_Whetstone(1);
      end loop;

      -- Assume that the task has completed this workload on time;
      -- if not, the count of met deadlines will be adjusted later

      Met_Deadlines := Met_Deadlines + 1;
      Next_Start    := Next_Start + Period;

    else

      -- A negative delay value means that either the workload was
      -- completed late (i.e. a deadline was missed, requiring load
      -- shedding to reset the task's next activation time) or that
      -- the load-shedding operation was somehow delayed long enough
      -- to cause the task to miss itss next scheduled activation time.

      if Met_Deadlines > Old_Met_Deadlines then
```

```
            -- A difference between the current number of missed deadlines
            -- and the last recorded value prior to load shedding indicates
            -- a missed deadline. Record the current missed deadline, adjust
            -- the met deadline count, and record the amount by which the
            -- task was late.

            Missed_Deadlines     := Missed_Deadlines + 1;
            Met_Deadlines        := Met_Deadlines - 1;
            Old_Met_Deadlines    := Met_Deadlines; -- Save until needed again
            Cumulative_Late      := Cumulative_Late + (- Next_Delay);

            -- Shed load by finding the current time (i.e. the time
            -- the last workload actually completed) and advancing the
            -- next starting time until it exceeds the current time,
            -- counting the number of deadlines skipped in the process

            Now := Next_Start + (- Next_Delay);
            while Next_Start < Now and Next_Start < Finish_Time loop
              Next_Start          := Next_Start + Period;
              Skipped_Deadlines   := Skipped_Deadlines + 1;
            end loop;

          else

            -- No difference between the current number of missed deadlines
            -- and the last recorded value indicates that while shedding load
            -- to catch up the task was delayed long enough to miss its next
            -- scheduled activation time. So, advance its activation time and
            -- skip one more deadline.

            Next_Start          := Next_Start + Period;
            Skipped_Deadlines   := Skipped_Deadlines + 1;

          end if;

        end if;

      end loop;

      -- Check to see if the final deadline was missed

      Next_Delay := Next_Start - Float( Calendar.Seconds(Calendar.Clock) );

      if Next_Delay < 0.0 and Met_Deadlines > Old_Met_Deadlines then
        Missed_Deadlines := Missed_Deadlines + 1;
        Met_Deadlines    := Met_Deadlines - 1;
        Cumulative_Late  := Cumulative_Late + (- Next_Delay);
      end if;

      -- Return the results

      Task_Met_Deadlines      := Met_Deadlines;
      Task_Missed_Deadlines   := Missed_Deadlines;
      Task_Skipped_Deadlines  := Skipped_Deadlines;
      Task_Cumulative_Late    := Duration(Cumulative_Late);

    exception -- Raised if Small_Whetstone fails its internal self-check
      when Workload.Workload_Failure => raise Task_Workload_Failure;

    end Do_Work;

    pragma Inline(Do_Work); -- Some implementations may ignore this
```

---------------------------oOo---------------------------

```
--|
--| Unit Name: T1 .. T5
--|
--| Unit Type: Task Body
--|
--| Description:
--|   Performs the requested workload at the given frequency.  The task
--|   will begin at the specified starting time and continue for the
--|   requested duration.  On completion, information concerning the
--|   ability of the task to perform the requested work is provided.
--|   to the calling program.
--|

task body T1 is

  Start_Time        : Calendar.Time;
  Length_Of_Test    : Duration;
  Frequency         : Float;
  Workload          : Natural;

  Met_Deadlines      : Natural;
  Missed_Deadlines   : Natural;
  Skipped_Deadlines  : Natural;
  Cumulative_Late    : Duration;

begin
  loop
    select
      accept Start (Test_Start_Time : in Calendar.Time;
                    Test_Duration   : in Duration;
                    Task_Frequency  : in Float;
                    Task_Workload   : in Natural) do
        Start_Time      := Test_Start_Time;
        Length_Of_Test  := Test_Duration;
        Frequency       := Task_Frequency;
        Workload        := Task_Workload;
      end Start;

      Do_Work (Start_Time, Length_Of_Test, Frequency, Workload, Met_Deadlines,
               Missed_Deadlines, Skipped_Deadlines, Cumulative_Late);

      accept Stop (Task_Met_Deadlines     : out Natural;
                   Task_Missed_Deadlines  : out Natural;
                   Task_Skipped_Deadlines : out Natural;
                   Task_Cumulative_Late   : out Duration) do
        Task_Met_Deadlines     := Met_Deadlines;
        Task_Missed_Deadlines  := Missed_Deadlines;
        Task_Skipped_Deadlines := Skipped_Deadlines;
        Task_Cumulative_Late   := Cumulative_Late;
      end Stop;

    or
      terminate;

    end select;
  end loop;
end T1;
```

------------------------------oOo------------------------------

```
task body T2 is

  Start_Time        : Calendar.Time;
  Length_Of_Test    : Duration;
  Frequency         : Float;
  Workload          : Natural;

  Met_Deadlines      : Natural;
  Missed_Deadlines   : Natural;
  Skipped_Deadlines : Natural;
  Cumulative_Late    : Duration;

begin
  loop
    select
      accept Start(Test_Start_Time : in Calendar.Time;
                   Test_Duration    : in Duration;
                   Task_Frequency  : in Float;
                   Task_Workload    : in Natural) do
        Start_Time      := Test_Start_Time;
        Length_Of_Test := Test_Duration;
        Frequency       := Task_Frequency;
        Workload        := Task_Workload;
      end Start;

      Do_Work(Start_Time, Length_Of_Test, Frequency, Workload, Met_Deadlines,
              Missed_Deadlines, Skipped_Deadlines, Cumulative_Late);

      accept Stop(Task_Met_Deadlines      : out Natural;
                  Task_Missed_Deadlines  : out Natural;
                  Task_Skipped_Deadlines : out Natural;
                  Task_Cumulative_Late   : out Duration) do
        Task_Met_Deadlines      := Met_Deadlines;
        Task_Missed_Deadlines   := Missed_Deadlines;
        Task_Skipped_Deadlines := Skipped_Deadlines;
        Task_Cumulative_Late    := Cumulative_Late;
      end Stop;

    or
      terminate;

    end select;
  end loop;
end T2;
```

---------------------------oOo---------------------------

```
task body T3 is

  Start_Time         : Calendar.Time;
  Length_Of_Test    : Duration;
  Frequency          : Float;
  Workload           : Natural;

  Met_Deadlines      : Natural;
  Missed_Deadlines   : Natural;
  Skipped_Deadlines : Natural;
  Cumulative_Late    : Duration;

begin
  loop
    select
      accept Start(Test_Start_Time : in Calendar.Time;
                   Test_Duration    : in Duration;
                   Task_Frequency  : in Float;
                   Task_Workload    : in Natural) do
        Start_Time      := Test_Start_Time;
        Length_Of_Test := Test_Duration;
        Frequency       := Task_Frequency;
        Workload        := Task_Workload;
      end Start;

      Do_Work(Start_Time, Length_Of_Test, Frequency, Workload, Met_Deadlines,
              Missed_Deadlines, Skipped_Deadlines, Cumulative_Late);

      accept Stop(Task_Met_Deadlines     : out Natural;
                  Task_Missed_Deadlines  : out Natural;
                  Task_Skipped_Deadlines : out Natural;
                  Task_Cumulative_Late   : out Duration) do
        Task_Met_Deadlines     := Met_Deadlines;
        Task_Missed_Deadlines  := Missed_Deadlines;
        Task_Skipped_Deadlines := Skipped_Deadlines;
        Task_Cumulative_Late   := Cumulative_Late;
      end Stop;

    or
      terminate;

    end select;
  end loop;
end T3;
```

------------------------------oOo------------------------------

```
task body T4 is

  Start_Time        : Calendar.Time;
  Length_Of_Test    : Duration;
  Frequency         : Float;
  Workload          : Natural;

  Met_Deadlines       : Natural;
  Missed_Deadlines    : Natural;
  Skipped_Deadlines   : Natural;
  Cumulative_Late     : Duration;

begin
  loop
    select
      accept Start(Test_Start_Time : in Calendar.Time;
                   Test_Duration   : in Duration;
                   Task_Frequency  : in Float;
                   Task_Workload   : in Natural) do
        Start_Time      := Test_Start_Time;
        Length_Of_Test  := Test_Duration;
        Frequency       := Task_Frequency;
        Workload        := Task_Workload;
      end Start;

      Do_Work(Start_Time, Length_Of_Test, Frequency, Workload, Met_Deadlines,
              Missed_Deadlines, Skipped_Deadlines, Cumulative_Late);

      accept Stop(Task_Met_Deadlines     : out Natural;
                  Task_Missed_Deadlines  : out Natural;
                  Task_Skipped_Deadlines : out Natural;
                  Task_Cumulative_Late   : out Duration) do
        Task_Met_Deadlines     := Met_Deadlines;
        Task_Missed_Deadlines  := Missed_Deadlines;
        Task_Skipped_Deadlines := Skipped_Deadlines;
        Task_Cumulative_Late   := Cumulative_Late;
      end Stop;

    or
      terminate;
    end select;
  end loop;
end T4;
```

------------------------------oOo------------------------------

```
task body T5 is

  Start_Time        : Calendar.Time;
  Length_Of_Test    : Duration;
  Frequency         : Float;
  Workload          : Natural;

  Met_Deadlines      : Natural;
  Missed_Deadlines   : Natural;
  Skipped_Deadlines  : Natural;
  Cumulative_Late    : Duration;

begin
  loop
    select
      accept Start(Test_Start_Time : in Calendar.Time;
                   Test_Duration   : in Duration;
                   Task_Frequency  : in Float;
                   Task_Workload   : in Natural) do
        Start_Time     := Test_Start_Time;
        Length_Of_Test := Test_Duration;
        Frequency      := Task_Frequency;
        Workload       := Task_Workload;
      end Start;

      Do_Work(Start_Time, Length_Of_Test, Frequency, Workload, Met_Deadlines,
              Missed_Deadlines, Skipped_Deadlines, Cumulative_Late);

      accept Stop(Task_Met_Deadlines     : out Natural;
                  Task_Missed_Deadlines  : out Natural;
                  Task_Skipped_Deadlines : out Natural;
                  Task_Cumulative_Late   : out Duration) do
        Task_Met_Deadlines     := Met_Deadlines;
        Task_Missed_Deadlines  := Missed_Deadlines;
        Task_Skipped_Deadlines := Skipped_Deadlines;
        Task_Cumulative_Late   := Cumulative_Late;
      end Stop;

    or

      terminate;

    end select;
  end loop;
end T5;
```

------------------------------oOo------------------------------

```
task body New_Task is

  Start_Time        : Calendar.Time;
  Length_Of_Test    : Duration;
  Frequency         : Float;
  Workload          : Natural;

  Met_Deadlines      : Natural;
  Missed_Deadlines   : Natural;
  Skipped_Deadlines  : Natural;
```

```
    Cumulative_Late    : Duration;

  begin
    loop
      select
        accept Start(Test_Start_Time : in Calendar.Time;
                     Test_Duration   : in Duration;
                     Task_Frequency  : in Float;
                     Task_Workload   : in Natural) do
          Start_Time     := Test_Start_Time;
          Length_Of_Test := Test_Duration;
          Frequency      := Task_Frequency;
          Workload       := Task_Workload;
        end Start;

        Do_Work(Start_Time, Length_Of_Test, Frequency, Workload, Met_Deadlines,
                Missed_Deadlines, Skipped_Deadlines, Cumulative_Late);

        accept Stop(Task_Met_Deadlines     : out Natural;
                    Task_Missed_Deadlines  : out Natural;
                    Task_Skipped_Deadlines : out Natural;
                    Task_Cumulative_Late   : out Duration) do
          Task_Met_Deadlines     := Met_Deadlines;
          Task_Missed_Deadlines  := Missed_Deadlines;
          Task_Skipped_Deadlines := Missed_Deadlines;
          Task_Cumulative_Late   := Cumulative_Late;
        end Stop;

      or
        terminate;

      end select;
    end loop;
  end New_Task;

end Periodic_Tasks;
```

```
--|-----------------------------------------------------------
--|
--| Unit Name:  Workload
--|
--| Unit Type:  Package Specification
--|
--| Description:
--|   Encapsulates the synthetic computational workload of a Hartstone task.
--|   The actual computation is performed by the Small_Whetstone procedure,
--|   a variant of the Whetstone benchmark program.  The amount of work
--|   requested is expressed in thousands of Whetstone instructions, or
--|   Kilo-Whetstones.  An internal consistency check is performed on the
--|   workload computation within Small_Whetstone; if it fails, an exception
--|   is raised.
--|
--|-----------------------------------------------------------

package Workload is

   Workload_Failure : exception;  -- Raised if Small_Whetstone self-check fails

   --|
   --| Unit Name: Small_Whetstone
   --|
   --| Unit Type: Procedure Specification
   --|
   --| Description:
   --|   Performs the computational workload of a Hartstone task.  The
   --|   computation is a scaled-down version of the one performed by the
   --|   full Whetstone benchmark program.  An exception is raised if the
   --|   computation fails to satisfy an internal consistency check.  This
   --|   procedure does not return any "result" from its computation; its
   --|   sole function is to give a Hartstone task something to do.
   --|
   --| Parameters:
   --|   Kilo_Whets: The number of Kilo-Whetstone instructions to be performed
   --|     by the procedure.  A value of 1 means one thousand Whetstone
   --|     instructions will be executed as the computational load.
   --|

   procedure Small_Whetstone(Kilo_Whets : in Positive);

   pragma Inline(Small_Whetstone);  -- Some implementations may ignore this

end Workload;
```

```
--|-----------------------------------------------------------
--|
--| Unit Name:  Workload
--|
--| Unit Type:  Package Body
--|
--| Description:
--|   See the description in the package specification and the description
--|   of the Small_Whetstone procedure below.
--|
--|   The Small_Whetstone procedure requires an implementation-dependent
--|   mathematical library.  Refer to the explanatory comments in the
--|   procedure for details.
--|
--|-----------------------------------------------------------

-- IMPLEMENTATION-DEPENDENT library name; see examples below
with Float_Math_Lib;
use  Float_Math_Lib;

package body Workload is

  -- IMPLEMENTATION-DEPENDENT subtype definition; see comments below
  subtype Whet_Float is Float;

  -- Instantiate the math library here, if necessary; see comments below

  -- IMPLEMENTATION-DEPENDENT library & function names; see examples in
  -- comments below
  function Log(X : in Whet_Float) return Whet_Float
    renames Float_Math_Lib.Log;

  --|-----------------------------------------------------------
  --|
  --| Unit Name: Small_Whetstone
  --|
  --| Unit Type: Procedure Body
  --|
  --| This version of the Whetstone benchmark is designed to have an inner
  --| loop which executes only 1000 Whetstone instructions.  This is so that
  --| smaller units of CPU time can be requested for benchmarks, especially
  --| real-time benchmarks.  The parameter "Kilo_Whets" determines the number
  --| of loop iterations; a value of 1 means the loop will execute 1000
  --| Whetstone Instructions. A Whetstone Instruction corresponds to about
  --| 1.3 machine instructions on a conventional machine with floating point.
  --|
  --| Small_Whetstone was developed by Brian Wichmann of the UK National
  --| Physical Laboratory (NPL).  The Ada version was translated at the
  --| Carnegie Mellon University Software Engineering Institute from the
  --| original standard Pascal language version (see references below).
  --| This Hartstone version has been adapted from the Ada standard
  --| version by making the Kilo_Whets variable a passed parameter, and
  --| by raising an exception, rather than printing an error message, if
  --| the benchmark's internal consistency check fails.
  --|
  --| Small_Whetstone uses the following mathematical functions, which are
  --| listed here using the ISO/WG9 Numerics Rapporteur Group proposed
  --| standard names for functions of a Generic_Elementary_Functions library
  --| (Float_Type is a generic type definition):
  --|
  --|   function Cos  (X : Float_Type) return Float_Type;
  --|   function Exp  (X : Float_Type) return Float_Type;
  --|   function Log  (X : Float_Type) return Float_Type; -- Natural logs
  --|   function Sin  (X : Float_Type) return Float_Type;
--|  function Sqrt (X : Float_Type) return Float_Type;
--|
--| The name of the actual mathematical library and the functions it
--| provides are implementation-dependent. For Small_Whetstone, the
--| function name to be careful of is the natural logarithm function;
--| some vendors call it "Log" while others call it "Ln". A renaming
--| declaration has been provided to rename the function according to
--| the ISO/WG9 name.
--| Another implementation-dependent area is the accuracy of floating-
--| point types. One vendor's Float is another's Long_Float, or even
--| Short_Float. The subtype Whet_Float is provided so that the change
--| can be made in a single place; users should modify it as necessary
--| to ensure comparability of their test runs.
--|
--| Examples of some vendor mathematical library and log function names,
--| and the values of the 'Digits attribute for the floating-point types
--| are provided in the comments below. The ONLY changes a user should
--| make to run Small_Whetstone are (a) the library name, (b) the log
--| function name, if necessary, and (c) the definition of the subtype
--| Whet_Float, if necessary. Any changes should be documented along
--| with reported results.
--|
--| References:
--|  The first two apply only to the full version of Whetstone. The
--|  first includes a listing of the original Algol version. The second
--|  includes an Ada listing. The third reference also deals mostly with
--|  the full Whetstone benchmark but in addition contains a brief
--|  rationale for the Small_Whetstone benchmark and a listing of its
--|  standard Pascal version.
--|
--|  Curnow, H.J., and Wichmann, B.A.
--|   A Synthetic Benchmark
--|   The Computer Journal, Vol. 19, No. 1, February 1976, pp. 43-49.
--|
--|  Harbaugh, S., and Forakis, J.A.
--|   Timing Studies Using a Synthetic Whetstone Benchmark
--|   Ada Letters, Vol. 4, No. 2, 1984, pp. 23-34.
--|
--|  Wichmann, B.A.,
--|   Validation Code for the Whetstone Benchmark
--|   NPL report DITC 107/88, March 1988.
--|   National Physical Laboratory,
--|   Teddington, Middlesex TW11 OLW, UK.
--|
--|------------------------------------------------------------


-------------------------------------------------------------
-- Math library for TeleSoft TeleGen2 VAX/VMS -> MC68020:
--
--   with Math_Library;
--
--   package Math is new Math_Library(Whet_Float);
--   use Math;
--
-- Natural logs (base e) = Ln(x); base 10 logs = Log(x).
-- There is also a pre-instantiated library called Float_Math_Library.
--
-- Float'Digits = 6; Long_Float'Digits = 15
-------------------------------------------------------------
-- Math library for Verdix VADS VAX/VMS -> MC68020:
--
--   with Generic_Elementary_Functions;
--
--   package Math is new Generic_Elementary_Functions(Whet_Float);
--   use Math;
```

```
--
-- Natural logs (base e) = Log(x);  base 10 logs = Log(x, Base => 10).
--
-- Short_Float'Digits = 6;  Float'Digits = 15
----------------------------------------------------------------------
-- Math library for DEC VAX Ada:
--
--   with Float_Math_Lib;
--   use  Float_Math_Lib;
--
-- Natural logs (base e) = Log(x);  base 10 logs = Log10(x).
--
-- Float'Digits = 6;  Long_Float'Digits = 15;  Long_Long_Float'Digits = 33
----------------------------------------------------------------------
-- Math library for Alsys Ada VAX/VMS -> MC68020:
--
--   with Math_Lib;
--
--   package Math is new Math_Lib(Whet_Float);
--   use Math;
--
-- Natural logs (base e) = Log(x);  base 10 logs = Log10(x).
--
-- If using the 68881 Floating-Point Co-Processor, the Math_Lib_M68881
-- package can be used.
--
-- Float'Digits = 6;  Long_Float'Digits = 15
----------------------------------------------------------------------
-- Math library for DDC-I Ada (DACS-80386PM) VAX/VMS -> i80386:
--
--   with Math_Pack;
--   use  Math_Pack;
--
-- Natural logs (base e) = Ln(x);  base 10 logs = Log(x, 10.0).
--
-- Float'Digits = 6;  Long_Float'Digits = 15
----------------------------------------------------------------------
-- Math library for Systems Designers XD Ada VAX/VMS -> MC68020:
--
--   with Float_Math_Lib;
--   use  Float_Math_Lib;
--
-- Natural logs (base e) = Log(x);  base 10 logs = Log10(x).
--
-- Float'Digits = 6;  Long_Float'Digits = 15;  Long_Long_Float'Digits = 18
----------------------------------------------------------------------

procedure Small_Whetstone(Kilo_Whets : in Positive) is

   T  : constant := 0.499975;  -- Values from the original Algol
   T1 : constant := 0.50025;   --   Whetstone program and the
   T2 : constant := 2.0;       --     Pascal SmallWhetstone program

   N8 : constant := 10;        -- Loop iteration count for module 8
   N9 : constant :=  7;        -- Loop iteration count for module 9

   Value     : constant := 0.941377;  -- Value calculated in main loop
   Tolerance : constant := 0.00001;   -- Determined by interval arithmetic

   I   : Integer;
   IJ  : Integer := 1;
   IK  : Integer := 2;
   IL  : Integer := 3;

   Y   : Whet_Float := 1.0;  -- Constant within loop
```

```
  Z   : Whet_Float;
  Sum : Whet_Float := 0.0; -- Accumulates value of Z

  subtype Index is Integer range 1..N9; -- Was a type in the Pascal version
  E1  : array (Index) of Whet_Float;

  procedure Clear_Array is
  begin
    for Loop_Var in E1'Range loop
      E1(Loop_Var) := 0.0;
    end loop;
  end Clear_Array;

  procedure P0 is
  begin
    E1(IJ)  := E1(IK);
    E1(IK)  := E1(IL);
    E1(I)   := E1(IJ);
  end P0;

  procedure P3(X : in Whet_Float;
               Y : in Whet_Float;
               Z : in out Whet_Float) is
    Xtemp: Whet_Float := T * (Z + X);
    Ytemp: Whet_Float := T * (Xtemp + Y);
  begin
    Z := (Xtemp + Ytemp) / T2;
  end P3;


begin -- Small_Whetstone

  for Outer_Loop_Var in 1..Kilo_Whets loop

    Clear_Array;

    -- Module 6: Integer arithmetic

    IJ := (IK - IJ) * (IL - IK);
    IK := IL - (IK - IJ);
    IL := (IL - IK) * (IK + IL);
    E1(IL - 1) := Whet_Float(IJ + IK + IL);
    E1(IK - 1) := Sin( Whet_Float(IL) );


    -- Module 8: Procedure calls

    Z := E1(4);
    for Inner_Loop_Var in 1..N8 loop
      P3( Y * Whet_Float(Inner_Loop_Var), Y + Z, Z );
    end loop;


    -- Second version of Module 6:

    IJ := IL - (IL - 3) * IK;
    IL := (IL - IK) * (IK - IJ);
    IK := (IL - IK) * IK;
    E1(IL - 1) := Whet_Float(IJ + IK + IL);
    E1(IK + 1) := Abs( Cos(Z) );


    -- Module 9: Array references

    -- Note: In the Pascal version, the global variable I is used as both
```

```
      --    the control variable of the for loop and an array index
      --    within procedure P0. Because the for-loop control variable
      --    of Ada is strictly local, this translation uses a while loop.

      I := 1;
      while I <= N9 loop
        P0;
        I := I + 1;
      end loop;


      -- Module 11: Standard mathematical functions

      -- Note: The actual name of the natural logarithm function used here
      --    is implementation-dependent. See the comments above.

      Z := Sqrt( Exp( Log(E1(N9)) / T1 ) );


      Sum := Sum + Z;

      -- Check the current value of the loop computation

      if Abs(Z - Value) > Tolerance then
        Sum := 2.0 * Sum; -- Forces error at end
        IJ := IJ + 1;     -- Prevents optimization
      end if;

    end loop;

    -- Self-validation check

    if Abs( Sum / Whet_Float(Kilo_Whets) - Value ) >
      Tolerance * Whet_Float(Kilo_Whets) then
      raise Workload_Failure;
    end if;

  end Small_Whetstone;

end Workload;
```

UNLIMITED, UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS NONE
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT APPROVED FOR PUBLIC RELEASE DISTRIBUTION UNLIMITED
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) CMU/SEI-90-UG-1	5. MONITORING ORGANIZATION REPORT NUMBER(S) ESD-90-TR-5

6a. NAME OF PERFORMING ORGANIZATION SOFTWARE ENGINEERING INSTITUTE	6b. OFFICE SYMBOL (If applicable) SEI	7a. NAME OF MONITORING ORGANIZATION SEI JOINT PROGRAM OFFICE
6c. ADDRESS (City, State and ZIP Code) CARNEGIE MELLON UNIVERSITY PITTSBURGH, PA 15213		7b. ADDRESS (City, State and ZIP Code) ESD/XRS1 HANSCOM AIR FORCE BASE, MA 01731
8a. NAME OF FUNDING/SPONSORING ORGANIZATION SEI JOINT PROGRAM OFFICE	8b. OFFICE SYMBOL (If applicable) SEI JPO	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F1962885C0003

8c. ADDRESS (City, State and ZIP Code)
CARNEGIE MELLON UNIVERSITY
SOFTWARE ENGINEERING INSTITUTE JPO
PITTSBURGH, PA 15213

10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.
	N/A	N/A	N/A

11. TITLE (Include Security Classification)
Hartstone Benchmark User's Guide, Version 1.0

12. PERSONAL AUTHOR(S)
Patrick Donohoe, Ruth Shapiro, Nelson Weiderman

13a. TYPE OF REPORT FINAL	13b. TIME COVERED FROM ______ TO ______	14. DATE OF REPORT (Yr., Mo., Day) March 1990	15. PAGE COUNT 90

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB. GR.

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The Hartsone benchmark is a set of timing requirements for testing a system's ability to handle hard real-time applications. It is specified as a set of processes with well-defined workloads and timing constraints. The name Hartstone derives from Hard Real Time and the fact that the workloads are presently based on the well-known Whetstone benchmark. This report describes the structure and behavior of an implementation in the Ada programming language of one category of Hartstone requirements, the Periodic Harmonic (PH) Test Series. The Ada implementation of the PH series is aimed primarily at real-time embedded processors where the only executing code is the benchmark and the Ada runtime system. Guidelines for performing various Hartstone experiments and interpreting the results are provided. Also included are the source code listings of the benchmark, information on how to obtain the source code in machine-readable form, and some sample results for Version 1.0 of the Systems Designers XD Ada VAX/VMS-MC68020 cross-compiler.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED XX SAME AS RPT. ☐ DTIC USERS XX	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED, UNLIMITED	
22a. NAME OF RESPONSIBLE INDIVIDUAL KARL SHINGLER	22b. TELEPHONE NUMBER (Include Area Code) (412) 268-7630	22c. OFFICE SYMBOL SEI JPO

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE. UNLIMITED, UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE